MASTER IN COMPUTER SCIENCE PROFESSIONAL FOCUS IN DATA SCIENCE

Evaluation and improvement of query performance and model visualisation in hybrid polystores

Marbehant, Sébastien

Award date:
2021

Awarding institution:
University of Namur

Université de Namur
Faculté d'informatique
Année académique 2020–2021

Evaluation and improvement of query performance and model visualisation in hybrid polystores

Marbehant Sébastien

Maître de stage : Kolovos Dimitris - Cleve Anthony

Promoteur : ______________________ (Signature pour approbation du dépôt - REE art. 40)
Cleve Anthony

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Contents

Grateful Acknowledgements

Working on the Typhon project... What a great adventure!

Like in every adventure, there was low and high. There were discoveries of peoples, technologies and methodologies. I want to thank all the people that have contributed to the redaction of this master thesis.

First of all, a special thanks to Pr. Cleve who has, since the first day, an open ear for my request. He has always supported me even in this complicated time of the COVID-19 pandemic.

I want to thank Pr. Kolovos for having overseen the project even in these evolving conditions.

Thank you to Pol Benats and Loup Meurice, who provided real support during all the project. From a seat in office, a place as a football player and insight about Typhon. They have helped to smooth the complexity of the project. They were also the professional and social link to the project.

For all the people from Namur, York or another place who took part in this paper, I show you my gratitude.

To my family, to Emilie, I will be forever grateful.

Abstract

The Typhon project aims to develop a methodology and technical infrastructure to support the graceful evolution of hybrid polystores, where multiple, possibly overlapping NoSQL and SQL databases may co-evolve consistently. This paper has for objective to improve the performance of the polystore by using a benchmark to analyse and make some recommendation and use visualisation tools to get keys information. Based on a threefold approach, we have made a literature review for each topic, bring a contribution and adapt the finding to the polystore environment, and created an implementation. The output of this research is a set of tools and recommendations to improve query performance and help users make the appropriate choice in designing the data model. Those tools could be integrated into the Typhon environment and help the user unleash the full potential of polystore.

Keywords: Polystore, Benchmark, Recommendation, Visualisation

Chapter 1

Introduction

1.1 Context

Times as flies, technology evolve quickly. There was a time where data banks were the must-have. From data banks to the relational database management systems(RDBMS), from the RDBMS to NoSQL database and now polystore, every paradigm change has brought new possibilities. This paper is concerned with the performance of polystore and, more specifically, Typhon polystore.

Since 70's the relational model has not evolved. Codd proposed the relational model to protect users against change in data representation. This model relies on a means of describing data with its natural structure only by well-known tables and relationships[7]. The principle based on the acronym ACID for Atomicity, Consistency, Isolation, and Durability was enunciated by Haerder and Reuter a few years later [17].

NoSQL comes from "non-SQL" or "non-relational", coined in a paper by Neal Leavitt. "Will NoSQL Databases live up to their promise?" was one of the first paper to compare RDMBS to NoSQL. The biggest advantage is the data model fit the data. ACID not compliance and scaling performance are evoked as advantages also.[26]

The database market has changed, "One size fits all" is no longer valid. The database must fit user needs. Inbound or outbound, aggregation or other primitives, the architecture of processing, availability, synchronisation are all criteria that could impact the database choice[38].

Many companies used multiple database management systems, but communication between database paradigms can be tricky and time-consuming. Polystore, also defined as polyglot persistence, is a solution who make paradigm co-exist to increase performance.[37]

User needs are the start of every product reflection and can drive product requirement. Multiple group of actors are competing on the market of hybrid polystore.The most well-knows project's name are BigDawgs and Typhon. The implementation of polystore can be from simple to really complex based on the coupling between database systems and implementation choice that are made by developers.

The BigDawg approach consists into storing the same data set into different data engine and use the data engine that provides the highest performance response to a particular query. The polystore has for purpose to seamlessly queried different data models from different database. Composed of several layers, figure:1.1, an interface layer has for purpose to communicate with the common interface/API, a island layer with an island for each database systems and underneath database systems. An island can interrogate its database system or another by using the shim operation. The cast operation is dedicated to do a data transformation from one data model to another. All those operation have for purpose to make the system appear to the user as a single entity.[4, 13]

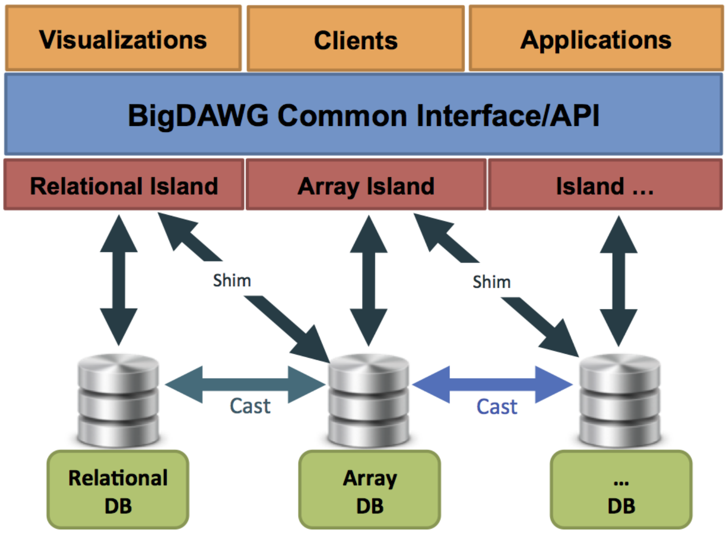

Figure 1.1: BigDawg Architecture

The Typhon approach is also based on a four layer architecture. The first is composed of multiple interface accessible by Eclipse plugging for the data model creation and generate the docker-compose script to run multi-container Docker applications or by using a browser and connect to local host addresses. The second layer is composed of a layer multiple container exchanging information and interconnected with the third layer composed of specific database containers. The last layer is composed of systems of records. Two languages have been developed to allow user to interact seamlessly with the different database paradigm. TyphonML to create a model and TyphonQL to query the model. The process behind the deployment can be seen in the figure 1.2.

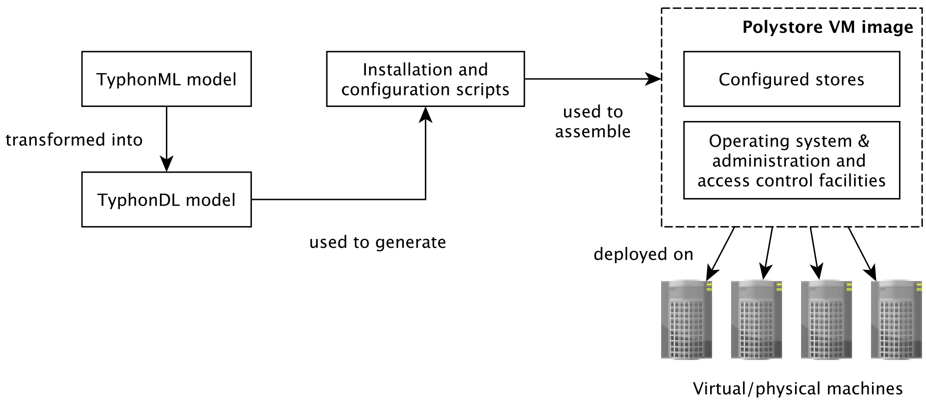

Figure 1.2: Typhon Deployment

BigDawg and Typhon have both the same purpose : allow user to interact with multiple databases paradigms but they do not have the same approach.

Criterion	Typhon polystore	BigDawgs
Own modelling language?	Yes	No
Own querying language?	Yes	Yes
Is/Has a middleware?	Yes	Yes
System of records based on a new paradigm?	No	No

Table 1.1: Comparison between BigDawgs and Typhon

This thesis tries to be as generic as possible but every contribution will be oriented towards improving the Typhon polystore and increase the user-friendliness of the systems.

1.2 Objective

One of the main aspect of the Typhon project is the continuous polystore evolution tools. This tool has for purpose to exploit the polystore query events captured by the monitoring mechanisms in order to recommend possible polystore schema reconfigurations (be they intra-paradigm or inter-paradigm). The goal of this Master's thesis is to contribute of the polystore evolution tools by proposing a consistent set of tools for continuous evolution of the hybrid polystore, for recommending polystore schema reconfigurations when relevant, and for informing the polystore evolution process. The recommendation will mainly focus on improving data access performance. The visualisation will improve user experience and help user to create meaningful queries.

1.3 Workflow

The workflow proposed, based on a literature review, the contribution is threefold. First, a performance analysis also know as benchmark is conducted. Second, recommendation to improve performance of one query or to improve the all set of query are proposed based on test performed with the benchmarking tools and existing recommendation in the literature. Finally with the help of visualisation technique a dashboard should be implemented to improve usability of the recommendation systems.

1.4 Methodology

This master thesis as for purpose to collect, transform and apply knowledge related to database management systems to polystore database toward a main goal, improving the performance. Polystore developments are new and this thesis try to explore main concern users and developers can have related to benchmarking, recommendation and visualisation of model.
The remainder of this thesis is structured as follows. The followings, literature review chapter presents findings about the three main topics, namely that they focus on benchmarking, recommendation and visualisation. Chapter 3 present how to evaluate performance of a polystore. This evaluation of performance is done by applying benchmarking technique into the polystore. Chapter 4 will use the knowledge

acquire to create recommendation to improve the performance of a polystore. Chapter 5 then helps the user to get back some understanding of the models by using a visualisation tools. The purpose of the visualisation tool is to let the user understands how the model is composed to help him applied change operators, modification of the model, or formulated some queries. Chapter 6 closes with a conclusion by resuming each chapter and starts discussion about interesting topics and future works.

Chapter 2

Literature review

From a open-eyed on three main topic, benchmarking, recommendation and visualisation, this literature review has for purpose to explore best practice in every domain and prepare them to a polystore context. The intend behind this literature review is to collect information to help us to contribute to the main goal : "Improve query performance". The benchmarking will focus on "How to make a database benchmark", the recommendation will gather information of how to improve specific database paradigm and the visualisation will focus on the creation of a visual to help user to create query.

2.1 Benchmarking

Conduct a performance analysis process is a complicated task. The first step is to explore common pitfalls to avoid making one during the Typhon benchmark. After that, we are going to explore the existing tools and hardware configuration. Data generation and queries are the following subjects to be studied the mimic reality. To end this literature review, we will explore the subject matter of how to expose the result.

One goal of this review is to know how to conduct a performance analysis of a database. This performance analysis should probe the complete system, also know as the macro-benchmark. On the opposite, micro-benchmark tends to explore a particular aspect of a system[18]. Another goal is to provide a solution to evaluate recommendation for the second part of this works. The benchmark has two roles: the first one to compare polystore against other database management system and the second one to provide a point of comparison between two queries optimisation strategies.

2.1.1 Common pitfalls

Commons error in performance analysis could be generalised to other sciences like the non-reproducibility and source of non-determinism of experience. More specific to computer science subjects like incorrect code, or more specific to database topic, optimises query and ignoring preprocessing time. This section will describe the common pitfalls which could appear during a benchmark.

Non-reproducibility

Every student who has one day took a science course must remember the first rule: "Every experiment must be reproducible". Computer science does not derogate from the ground rule. Configuration parameters and source code should be available[35]. Experiments can not always be fully reproducible (e.g. due to legal concern). It is then important to be able to characterise the level. The depth is how much the experiment is made available or archived. Another concern is portability: can results be reproduced in the original environment or a different one? The last concept is the coverage, partial or full reproducibility of the experiment.[12]

Ignoring noise and source of non-determinism

The Source of non-determinism can be the system, the application or the environment[19]. Comparison between two database paradigm can only be fair if the same query is performed on each system. Make a comparison between a full-fledged DBMS and a standalone program could lead to uneven result due to queries parallel processing or another process.[35] Flushing cache[1][2] and restart database is needed to conduct cold run. It is not always possible, especially in a cloud environment. The most important aspect to cold, warm and hot runs is to avoid comparison[19, 35].

Incorrect code

Due to development, a bug in implementation could still occur during the performance analysis. The correctness of the result should be the primary concern.[35]

Optimisation

We need to do the optimisation with care in performance analysis. We need to configure each DBMS to reach a top performance to produce a fair comparison, but optimisation should never improve performance in a specific benchmarking test. Maximise the system's performance by database optimisation should be done correctly. One usage of the benchmark is to compare two solutions. One is often the author algorithm or technique compared to an existing system. That could lead the author to poorly optimise the state of the art to make his solution look better than another. One recommendation is to follow guidelines or to explore configuration used in previous publication.[35] Standardised benchmark specifications could lead to optimisation toward a specific benchmark[35] or benchmarketing when benchmark a create to fit DBMS for marketing reason[31].

Ignoring preprocessing time

Indexing, a prepossessing task, takes time to construct and, depending on the DBMS, could lead to a significant difference in performance. The construction of preprocessing should be taken into consideration if preprocessing are not available in every system.[35]

[1] echo 3 > /proc/sys/vm/drop_caches with root privileges on linux systems
[2] sync && sudo purge on macOS

2.1.2 Benchmarking tools

TPC Benchmark and Yahoo! Cloud Serving Benchmark (YCSB) are the most well know benchmarking tools. The existing solutions are not suited for Typhon Polystore. Unit against benchmarketing techniques (running tailored benchmarks to increase sales), a small group of individuals contributes to creating a fair and neutral means to compare performance. The Transaction Processing Performance Council (TPC) is composed of influential academic database experts and well-known industry leaders[31]. TPC does not provide an easy parametrizable framework to assess the performance of a database. YCSB framework was created by Yahoo! to provide a comparison solution for cloud database. The framework evaluates a database based on the cloud needs like scale-out, elasticity and high availability, which are not the primary concern of Typhon polystore. YCSB provide an open-source solution with a set of workloads specified with principal features (Insert, Update, Read and Scan) which could inspire our solution[8].

2.1.3 Experimental setup - Hardware configuration

The easiest way to make the experience reproducible is to facilitate other people to conduct the same benchmark by providing the configuration. Ceesay et al. have proposed a solution to make benchmarking easier to set up by containerising the tools and settings. Two main components are the plug functional and the play functional. They are respectively in charge of eliminating the manual processes by auto-detecting settings, adding them to the configuration files, and running the benchmark.[5] It is recommended to fix the environment with slight variation as possible to focus on a specific factor. Depending on the environment fixing all the parameters may not be possible. It is then recommended to change the execution order randomly. The minimal description should include software (and version), operating system version, and compiler flags. Depending on the relevance, should include details of the network, the connection bus, or the main memory bandwidth[19]. Even in a fixed environment changing an innocuous aspect of an experimental setup could lead to bias. Mytkowicz et al. pointed out that a simple change as the UNIX environment size could lead to a change in performance. Make an exhaustive list of bias is not a solution to avoid them in performance analysis. Learn to deal with it is a better solution. A solution to reduce bias is to generate many experimental setups and then use statistical tools to compare results. To detect if an incorrect conclusion was reached from our data, we could use causal analysis. This solution does not exclude bias totally but tends to reduce them. [30]

2.1.4 Data generation and Queries

The last concern to evaluate a database performance are the generation of data and query. We are going to analyse the tools used by well-known benchmark and independent tools.

Each benchmarking tools has its generator or its solution to populate the database. For example, TCP-H speaks about "The queries and the data populating the database have been chosen to have broad industry-wide relevance."[40] . TCP-C define a database schema in his documentation.[39]. YCSB provide a workload and allow the user to create its own if needed[15]. The workload characteristics(proportion of: create, read, update and delete query) allow testing a broad sort of user need[8].

Independent workload generator like NoWog tries to fill the void left by traditional benchmarking tools. NoWog is a workload generator based on a specific grammar that is independent of a particular NoSQL database paradigm. Synthetic workloads are developed to have similar characteristics to real workloads and test the three majors indicators: volume, velocity and variety[1] (extended with value and veracity [10]).

2.1.5 Exposing the result

Making a performance analysis without falling into common pitfalls is not all. The last step is to expose the result without changing the reality. We should consider a set of common-sense rules in each paper. The first is to report the unit unambiguously and always use the official notation(e.g. Bytes with B and Bits with b). The second rule is to provide all the result and not only the ones with the desired effect. Taking special care to use summarising ratio with parsimony and report if the measurement is deterministic. The last tips are to plot as much information as possible, this recommendation will allow users to compare the result with our conclusion. [19]

2.1.6 Expected criteria for a benchmark

In the following section, special care will be provided to avoid common pitfalls previously cited. We have defined a list of criteria to respect (table: 2.1). We have attributed importance to each rule based on the following nomenclature.

- Req. for Required: Minimum criteria to conduct a performance analysis.
- Exp. for Expected: Give additional information to the user. This information can facilitate the process of reproducibility.
- Opt. for Optional: Information that helps to explain the context but not mandatory for the reproduction.

Criterion	**Details**	**Req.**	**Exp.**	**Opt.**
Hardware description	Processor Model / Accelerator RAM Size / Type / Bus Info NIC Model / Network Infos	✓		
Query description	List of query	✓		
Model description	Complementary information on the model	✓		
Database description	Version and Configuration	✓		
Optimisation	Optimise each query	✓		
Avoid Bias	Large number of experimental setups	✓		
Detect Bias	Causal analysis		✓	
Set of query	CRUD(Create, Read, Update, Delete)		✓	
Dataset	Data from the database		✓	
Database Images	Container with database image			✓

Table 2.1: criteria for a benchmark

2.2 Improving a database model

Improving the performance of query performed on a database can be done based on different strategies. Even the most simple database can be decomposed into two-layer at the highest level of abstraction, a physical layer composed of disk storage and database management software. On top of those layers, we have the programming language, which interrogates the hardware based on the translation of commands.

2.2.1 Database performance optimisation by hardware tuning

The hardware evolve at a high pace due to new discovery or linked to the Moore's law. Database solution providers, like Oracle, also propose some warranty or evolution in they contract. This evolution and warranty make the hardware on of the easiest solution to increase performance of a database management systems.
When a company needs better performance for the database system, it is easier to improve the hardware. This modification of hardware can be after finding the bottleneck from changing a disk, adding some RAM, changing the CPU, etc.
In their articles, J. Cieslewicz and K. A. Ross have explained the evolution of hardware for database management systems and present some change possible towards modern architecture. As they have said: "Database optimization for modern hardware is an area of research where the work is never finished".[6]

2.2.2 Database performance optimisation by software tuning

Improving query performance based on modification into the application layer is usually not the primary concern of companies due to the task's complexity. Usually, database systems are tuned in by creators or by a community of developers in open source.

2.2.3 Database performance optimisation by the data model and query optimisation

Improving the data model or query to be more efficient is usually done based on best practice or tests. Database management systems provide solutions like an index to increase a query's performance and solution provided by best practice like use the most precise type when you defined a field in an SQL database. Every improvement can have disadvantages, and it is usually a matter of compromise. We need to improve the global performance of the polystore. We will enhance each query to reduce kink in query formulation and then have a global approach to optimise queries.

One query at a time Improving the performance of a query is not always possible. The formulation of the query or the data models can be improved. First of all, the data models must be examined. Is it an index? Are the type of data well adjusted? The second concern is the query, and some tools can be used to review the execution plan.

Multiple query at a time NoSQL paradigms are easier to scale. There are used because the number of request on a system is high. The increase of request on a database is due to the rise in usage of systems. Today, a database is more queried, and those queries have different shapes. Taking into account the number of occurrence of each query has more sense but is not always possible. Each improvement on one query can have disadvantages on another one. Adding an index on a database that is most of the time used to perform "create" queries will negatively impact the global performance.

2.2.4 How to improve a specific language?

MariaDB - Relational

In the relational paradigm, we can categorise query based on the type of CRUD operations or based on the query's complexity depending on the structure. If a query interrogates one table, we can say that a simple query. If a query interrogates two or more tables, we can say that it is a more complex query. The easiest way to increase the performance of a simple query is to create an index on the where clause's attribute. For more complex queries, indexing can be a solution, but the order of the argument in a query could also vary the performance.

Indexing There are four possibles types of index in a relational database: primary keys (unique and not null), unique indexes (unique and can be null), plain indexes (not necessarily unique) and full-text indexes (for full-text searching).[14]
It is recommended to create an index to match the queries based on the application's needs. Any extra will waste resources. Using the "EXPLAIN" statement can help to decide which columns need indexing. If there are queries that contain the keyword "Like", it is also recommended to use a full-text index. Removing a rarely used index could increase INSERT and UPDATE performance(The information Schema INDEX_STATISTICS provide info on the index's usage).[14]

Joins Joins two or more tables in a query could take some time. SQL is a declarative or non-procedural language. Those types of languages specify what is to be done rather than how to do it. [9] In this case, we ask for specific data from a specific location. The particularity of a declarative language is that the database management system handles a query execution plan. Thus, we have almost no possibilities to modify performance.
Leis et al. have explored how database management system deals with joins queries and how to improve performance using different estimations like cardinalities. [27]

Types Using the most precise type for an attribute can increase the performance significantly. For example, storing an uniqueID composed of a numeric value into a String is nonsense. [33]

Cassandra - Key-value database

Key-value databases are from the NoSQL family. In this subsection, we are going to explore the recommendation provided for data modelling. As Typhon used Cassandra, we are going to explore recommendation for this database. Those recommendations could slightly differ from one database to another database, but principles are similar.

Data Modeling Data modelling is query-driven and implies starting from the query to create a database structure. There is no join in Cassandra. All the information needed must be at the same table. The last big difference with a relational database is the duplication of data across multiple tables. Choosing the right primary and partition key is fundamental to have a good performance. The syntax used to create a primary key is essential. In the example below, the primary key is composed of three arguments. The first is the id and is called the "partition key", the second and last argument is "clustering key". Partitioning is done by the id, and in the partition, rows are ordered by "productName" and "suppliersId". The partition key can be composed of multiple arguments grouped using parentheses. Choosing the right partition key is essential. By reducing the number of partition read, the performance is improved.

With the last release of Cassandra (4.0) materialised view was implemented but are still experimental. Thus, this solution does not take the materialised view into account. In a latter development, MV could reduce disk space needed as the view tables are constructed from data in another table called the base table. [21]

```
CREATE TABLE Products (
    id uuid,
    productName text,
    quantityPerUnit text,
    unitPrice float,
    unitsInStock int,
    unitOnOrder int,
    reorderLevel int,
    discontinued boolean,
    categoriesId text,
    suppliersId text,
    PRIMARY KEY (id, productName, suppliersId)
) WITH CLUSTERING ORDER BY (productName DESC)
```

Neo4J - Graph database

One of the most used graph databases is Neo4J. As Typhon used it, we will explore what is possible to do and how to increase the performance. This paper will only take the modelling aspect because Typhon handles the rest.

The Neo4J's documentation has a complete chapter on performance improvement. Two mains aspects are index and create a graph that corresponds to the queries executed on the database.

Index In Neo4J, there are two different types of index. The first one is full-text. Those are optimised for indexing and searching text. The only way to access the index is by being queried explicitly via procedures. The second type is the b-tree. Those could be created and dropped using Cypher. Cypher's query planner integrated b-tree index when needed. [20]

Graph Modeling As with NoSQL, the first aspect to take into account is the queries. Queries could highly impact performance. For example, a client who lives in the UK. If we have a query that looks for all the clients who lived in a country, we should create a node for the UK with a label country and compare the client nodes and the country nodes. If the only access one client and want is country. We should make a property named "Country". With a relatively small dataset, the impact could be trivial but not with a big dataset.[2, 16]

MongoDB - Document database

MongoDB is the last database paradigm where are going to explore. MongoDB is a NoSQL and document database. Like the two NoSQL databases previously studied, it is recommended to start from the queries to model the database. MongoDB documentation has a large section dedicated to performance. Each of the following paragraphs develops a recommendation. We have already reduced recommendations to the limits of Typhon.

Data modeling and sizing memeory Data with a 1:1 relation should be modelled in a single document(embedding). This choice induces better performance for read operations. Referencing should be used when

- A document is frequently read but contains data that is rarely accessed. Embedding this data increases only the in-memory requirements (the working set) of the collection.
- One part of a document is frequently updated and constantly growing in size, while the document's remainder is relatively static.
- The combined document size would exceed MongoDB's 16MB document limit, for example, when modelling many:1 relationships, such as product reviews to a product. [25]

Index There are seven types of index in MongoDB. Regular index for matching the entire value of a field (1). Compound index for index composed of multiple attributes (2). Text index to match a specific word in a text (3). Wildcard index for supporting queries against unknown or arbitrary fields (4). Partial index for only indexing document that succeeds a conditions(5). Multi-key index: to index each element of an array (6). Case insensitive index: to avoid the use of regex for case insensitivity (7). [23]

Multi-document Transaction The multi-document transaction appears with version 4.0. As MongoDB was not previously designed to handle this scenario, some limits must be respected. By default, the transaction runtime limits are 60seconds, and it is recommended to modify less than 1000 documents per query.[24]

2.3 Visualisation of a database model

The goal is to create a good visualisation tool to help the user executes efficient queries on the polystore. There are two main aspects to consider: the first is the interaction between the user and the polystore, and the second is about the visualisation. This section will explore the possibilities and best practice to create visualisation.

2.3.1 How to design an interface with guidelines?

Even exposed through a navigator, a website is an interface also now as a web user interface. It can be defined by: "A Web user interface or Web app allows the user to interact with content or software running on a remote server through a Web browser. The content or Web page is downloaded from the Web server, and the user can interact with this content in a Web browser, which acts as a client."[32].
There are a lot of guidelines to design an interface based on your support. For example, Apple[11] and Microsoft with fluent[28] have defined the best guidelines to design an interface on their operating systems. There are web standards developed by the W3C organisation.[45]

2.3.2 How to create a memorable and positive user experience with the polystore?

My first contact with the polystore left me with a strange feeling of powerful but not usable technology. This first interaction made me think about the problem of interaction between human and machine. The experience was not designed for users. The definition from the UX Book for user experience is a good start to think about the future interface that needs to reconcile user and machine: ≪ User experience (UX) is the totality of the effect or effects felt by a user as a result of interaction with, and the usage context of, a system, device or product, including the influence of usability, usefulness and emotional impact during interaction, and savoring the memory after interaction. "Interaction with" is broad and embraces seeing, touching, and thinking about the system or product, including admiring it and its presentation before any physical interaction. ≫[34]
With the user experience definition in your minds we can start to analyse, design, implement and evaluate the interface.

2.3.3 What-Why-How model

To satisfy the goal, it is easier to use framework. The What, Why, How is a three-part analysis framework used in visualisation and more precisely in data-viz. "Why is the task being performed, what data is shown in the views, and how is the vis idiom constructed in terms of design choices".[29]

Chapter 3

Evaluation of performance

Improving data access performance requires first evaluating the performance. Benchmarking techniques are well used in a single paradigm database to extract valuable insight and get a landmark between all the technology available on the market. Those techniques are the first stone to helps us to define less efficient structures, formulation into the data model or queries. The remainder of this chapter is organised as follow Composed of five-section, this chapter leads us to solve the problem of conducting a benchmark in a polystore environment. After a brief introduction, the contribution will be based on the knowledge acquired during the literature review, an implementation is provided, tested, and results are exposed. The conclusion review the work and make recommendations to improve the document.

3.1 Introduction

In the Cambridge dictionary, a benchmark is "a level of quality that can be used as a standard when comparing other things"[3]. The literature review provides us with a methodology and a list of expected criteria to satisfy the minimum requirements to do a qualitative performance evaluation (figure 2.1). We have defined three levels of necessity for criterion from optional to required. The contribution will examine each of the criteria and translate those criterion to the polystore environment. The implementation will test the procedure defined in the contribution.

3.2 Contribution

The performance analysis process of Typhon as for goal to state rules on how to take advantage of the hybrid polystore. Each paradigm, relational, graph, key-value, document as his advantages and inconveniences. By producing multiple models and make the comparison between queries result, we can extrapolate recommendation. This contribution is twofold. The first one is about hardware and software configuration. The second one is about the data model, queries and retrieves result.

3.2.1 Hardware and software configuration

It is essential to get a complete description of the environment in which tests are performed to allow complete transparency, reproducibility and the ability to understand the result. We have divided the description into three-part. The first describes the computer hardware and software; the second is about Docker, and the last about Typhon. We have provided an extensive description in the appendix. B

Computer

We conducted our experiments on a MacBookPro16,1 composed of one single 6-Core Intel Core i7 of 2.6 GHz with 256 KB L2 Cache per Core and 12MB as L3 Cache. The computer has 16 GB of RAM (2*8GB - DDR4 - 2667 Mhz) and an APPLE SSD 500GB. The version of OS is macOS 10.15.7 (19H2), and the kernel version is Darwin 19.6.0.

Docker

Typhon required Docker to launch the polystore. The version of Docker is v19.03.13. To increase the polystore performance, Docker has access to 6 CPUs, 7.00GB of Memory, 1GB of Swap and 59.6GB for the disk image.

Typhon

The Typhon polystore environment is composed of multiple images. Each image has a unique ID. Image ID can be found in the appendix. (B)

3.2.2 Data model, queries and retrieve result

The database needs a model and data to fill it to allow a user to perform queries. The benchmarking process use performance of queries to compare model, database paradigm or DBMS. Each of these particularities can influence the final result. Our role is to reduce bias induced by the process to be as close to reality as possible.

3.2.3 Data and model

Create artificial data has the disadvantage of not representing reality or being incomprehensible. The goal of the performance analysis is to match the reality. The easiest way to get close to reality is to use actual data. Disadvantages are multiple. Data are sometimes under copyright, highly protected by the company due to business secret or by law to protect consumers (e.g. GDPR).
For experimental concern, slow evolution and focus on one element could teach us more than a global picture. The model used was a part of the well knows Northwind database. It is a sample database used by Microsoft to allow a user to test database management systems. The Northwind database is strongly distributed and available on multiple DBMS like Neo4J[41] or MariaDB[36]. The model of Northwind, figure 3.1, represents fictitious company that imports and exports speciality foods from around the world. For testing purpose, migrating the entire database from one concept to another and fixing the number of change to one to be able to extract valuable insight was not possible. We have extracted from the Northwind model two sub model.

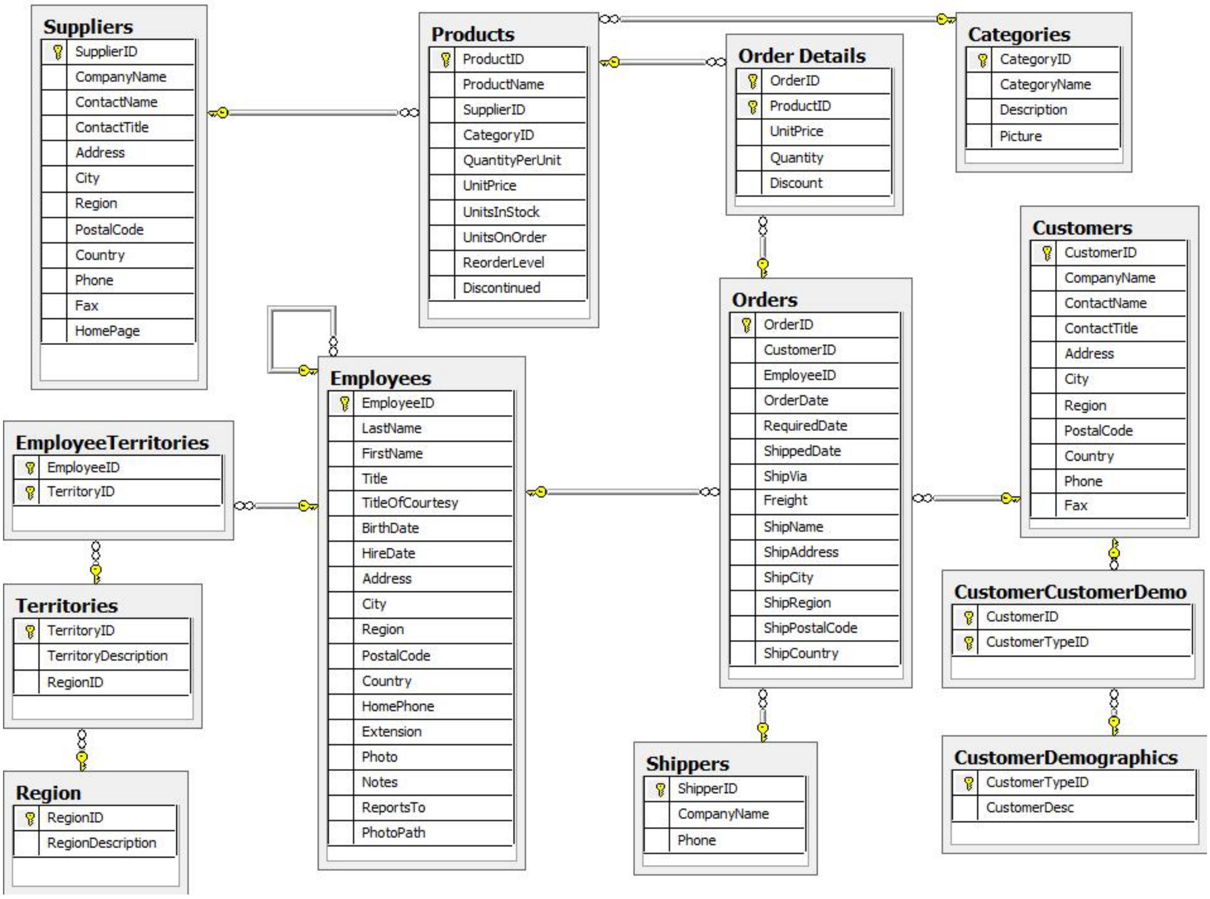

Figure 3.1: Northwind Diagram

Model 1:

The first model, figure 3.2, is composed of one entity called ”Products” without relation. This entity will be migrated between each paradigm with its data.

The corresponding syntax in TyphonML:

```
entity "Products" {
    "ProductId" : int
    "ProductName" : string[40]
    "QuantityPerUnit" : string[20]
    "UnitPrice" : float
    "UnitsInStock" : int
    "UnitsOnOrder" : int
    "ReorderLevel" : int
    "Discontinued" : string[3]
    "CategoriesID" : int
    "SuppliersID" : int
}
```

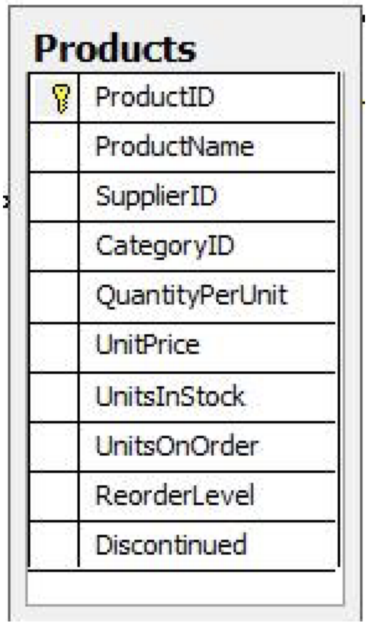

Figure 3.2: Model 1 - Products entity (From Northwind Diagram)

The migration tools will relocate entities and data from one database to another supported by Typhon with change operators. For example, the change operator used to migrate the database product to a key-value database.

```
changeOperators [
    migrate Products to KeyValueDatabase
]

```

Model 2:

The second model, figure 3.3, comprises two entity ”Products” and ”Suppliers”, with relations between them. Those entities will be migrated one by one in each paradigm with the related data.

```
entity "Products" {
   "ProductId" : int
   "ProductName" : string[40]
   "QuantityPerUnit" : string[20]
   "UnitPrice" : float
   "UnitsInStock" : int
```

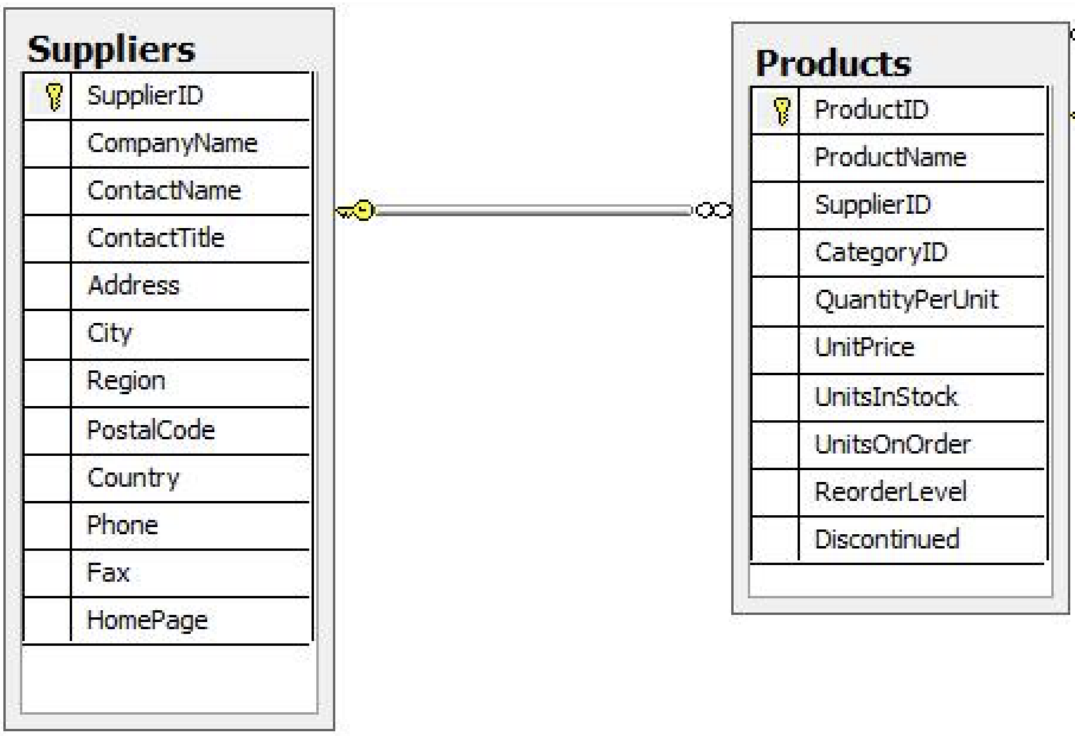

Figure 3.3: Model2 - Products and Suppliers entity (From Northwind Diagram)

```
    "UnitsOnOrder" : int
    "ReorderLevel" : int
    "Discontinued" : string[3]
    "CategoriesID" : int
    "SuppliersID" : int
    "Suppliers" -> "Suppliers"[1]
}

entity "Suppliers" {
    "SuppliersId" : int
    "CompanyName" : string[40]
    "ContactName" : string[30]
    "ContactTitle" : string[30]
    "Address" : string[60]
    "City" : string[15]
    "Region" : string[15]
    "PostalCode" : string[10]
    "Country" : string[15]
    "Phone" : string[24]
    "Fax" : string[24]
    "HomePage" : text
```

```
    "Products" -> "Products"."Products.Suppliers"[0..*]
}

```

Queries

Each user or type of user has a favoured type of query. Depending on the need, the performance of a database could change. If the user tries to understand his business, he will make more read action than write. To match each user profile, we have defined three batches of queries(3.1). The first one, named Type 1: ”Heavy create and update” is composed of a majority of create and update operations. The syntax to formulate an insert or read query is:

```
insert TableName {attributes : value (, attributes: {value)* }
```

```
update TableName Alias
where condition
set {attribute:value(,attibute:value)*}

```

This syntax in the Northwind context can give for example:

```
insert Categories {
@id:#GenHRCat ,
CategoryName : "gen Category"
Description : "This is a gen Category"
}
update Products p
where p . ProductName =="Chai" set {price :+ 5}

```

The second type, named Type 2: ”Heavy read” is composed of a majority of read operations. The syntax and one example can be found bellow.

```
from TableName Alias(,TableName Alias)*
select Alias(.Attribute),(Alias(.Attribute))*
where Condition

```

```
from Orders o , Shippers s
select o.OrderDate , o.@id , o.OrderDate , o.RequiredDate , s.
CompanyName , s . Phone
where o== s . Orders && s .CompanyName == "Speedy Express"

```

The last type is named Type 3: ”Balanced” is composed of the same quantity of creade,read and update operation and of 10% of delete operation. This leads us to the syntax and an example of a delete query:

```
delete TableName Alias
where condition

```

```
delete Customers c wherec.@id==#GenHCCons

```

Type	Create	Read	Update	Delete
Type1 : Heavy create and update	44%	6%	44%	6%
Type2 : Heavy read	14%	70%	10%	6%
Type3 : Balanced	30%	30%	30%	10%

Table 3.1: Distribution of type of query by user profile

An extended version of the syntax TyphonQL and all the queries used can be found in appendix D.

Result

The result can be seen at different levels. The lowest level, where each query are taken independently. The higher level where queries results are aggregate based on the type (CRUD) or based on a batch. The Typhon project provide multiple access to result from the so-called post-execution events (PostEvent), from the continuous evolution tool or from the benchmark java program. Respectively, raw data, a more user-friendly approach and system time output.

Each query individually The first aspect to consider is to make recommendations based on single query performance to improve model performance. Post-Event, continuous evolution tool and java program are explored to find the best way to improve a single query.

PostEvent PostEvent are event-generated by the TyphonQL engine and published with Apache Kafka's help and process with Apache Flink. We could retrieve raw execution rapport by subscribing to the PostEvent queue.[1]

Continuous evolution tool The continuous evolution tool provides visual analytics of Polystore data usage. This visual tool as the advantage to retrieve result for each query and retrieve the result from similar queries with the help of wildcard.

Java program: Benchmark By capturing in the response header of a query the "QL-Wall-Time-Ms", a java program is capable of getting the exact result for each query.

Batch of queries It is essential to take a step back to get a global view to increase the database performance. Enhancing a single query could lead to decreasing global performance. Post-Event, continuous evolution tool and java program are explored to find the best way to retrieve result for a batch of queries.

PostEvent Aggregation of PostEvent could be a solution to retrieve execution time for a batch of queries. Due to the complexity of this solution, it has not been implemented.

Continuous evolution tool The continuous evolution tool does not provide information on the processing time for a batch of queries.

Java program: Benchmark The information provided by the java program benchmark's execution make by summing each query's result is the easiest solution for batch queries.

3.3 Implementation

Each query will be built on top of one or multiple database paradigms to compare the performance of Typhon. The Typhon polystore is configured to be used with MariaDB, MongoDB, Cassandra and Neo4J. Respectively relational database, document database, key-value database and graph database. This section will describe each model and how it could be implemented in a Typhon polystore. Due to migration problems towards Cassandra and Neo4J this part focus on Relational and Document Database.

3.3.1 How? The benchmarking procedure

1. Polystore is created with the model developed in the contribution: data and model 3.2.3.
2. The polystore is built with docker.
3. The polystore is reset (To create databases structure)
4. The polystore is filled with data with the data_ingestion tool provided by the Typhon evolution tools.
5. Query are executed with the java program developed for this purpose.

[1] Architecture description could be retrieved in the deliverable 5.3 Event Publishing and Monitoring Architecture

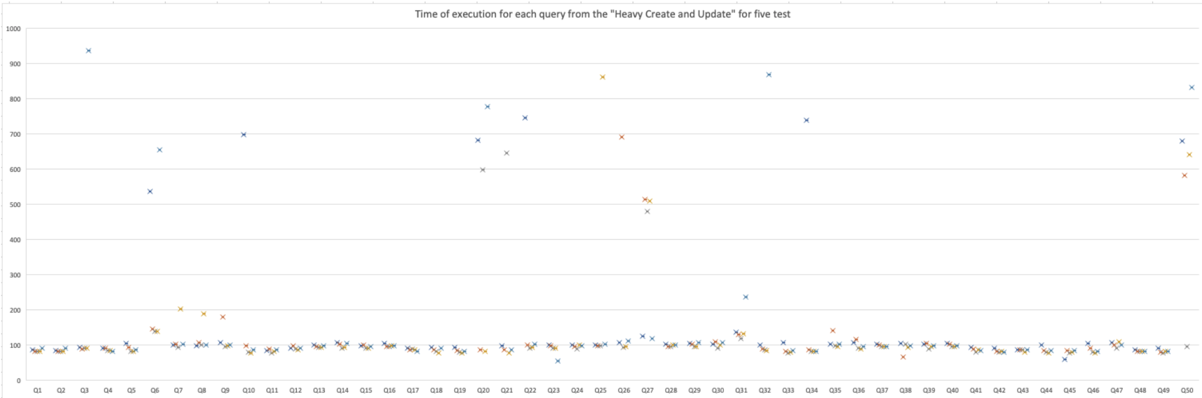

Figure 3.4: Benchmark result: Heavy Create and Update

6. Cold results are recorded.
7. The databases are reset (To empty the databases)
8. The polystore is filled again.
9. Queries are executed again (Warm test)
10. Warm results are recorded.
11. The polystore is deleted with docker-compose rm -v as recommended in the Typhon documentation.

This procedure is repeated five times.

3.3.2 Queries performed

Based on the profile defined in 3.1 a list of queries have been defined in 3 files. Each file contains 50 queries which are executed by a Java program called "Benchmark". The process is composed of 4 steps. Extract query from .tql files (1), transform query to match API recommendation (2), execute query one by one (3) and Return time of execution (4).

3.3.3 Result

We have created a model as simple as possible without performance tuning (no index, ...) to have a point of comparison. The results extracted from the benchmark are of two types: the white and black box. The choice to keep white box result is to have the maximum of information to explain the variation of results. Even if the white box results are present, we should only consider the black box results because a user consider only the time he need to get the result and the purpose of this project is to be independent of Typhon.

Type 1 (figure 3.4), Type 2 (figure 3.5) and Type 3 (figure 3.6) results are exposed in three figure bellow. Each exposed the result of 50 queries run five times on the polystore.

Figure 3.5: Benchmark result: Heavy Read

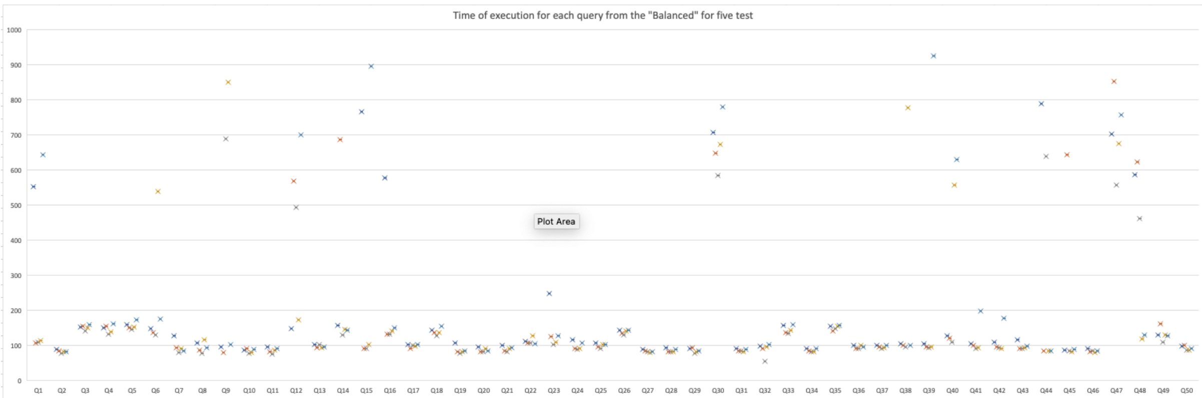

Figure 3.6: Benchmark result: Balanced

3.4 Conclusion

This methodology could be improved by running the queries on a different setup and see if similar results can be observed. The number of queries could also be increased to reduce bias. For that purpose, a query generator could be implemented. A Jenkins's test server could also be implemented to improve performance and reduce manual task. The complexity of creating, launching, deploying a polystore, making a model evolve, or understanding an error has not eased this procedure's automation. For example, It is always necessary to create a model manually or look in the log to understand why a query failed because the interface only responds with an "Error". With this technology's evolution highly desired by the public and private sector, this benchmarking process could be smooth out.

Chapter 4

Recommendation

Improving data access performance requires making some change to the data model or query once we have acquired more knowledge by testing the model in a benchmarking, for example. Change can be recommended to a user based on his habit or need. The term ”recommendation” is used to describe every modelling choice or query adaptation that could improve the performance of the polystore. The performance criteria are multiple and could be the execution time of one or multiple queries. The remaining of this chapter is organised as follow. After a brief introduction resuming the literature review and purpose of this chapter, we will start to contribute to the model evolution, then we will test the contribution on a Typhon polystore and draw a conclusion from the experimentation.

4.1 Introduction

In the previous chapter, we have created and tested models. The models can be relatively slow for some queries. There must exist improvement to perform on the data models to increase the performance of queries. Those improvements are recommendation and, combined with the best practice from the literature, will be the foundation of a set of rules to improve one query or a list of queries. Every performance improvement can lead to disadvantages. Those disadvantages could be from multiple types like increasing disk space needed for indexing, downgrading another query's performance for splitting an entity, or time-consuming for migrating database. We will define for each recommendation the impact positive or negative on the polystore.

4.2 Contribution

In the previous section, we have searched for the best practice already well-known by the database community. Those best practice could lead to the enhancement of performance in a polystore environment. Based on the previous chapter, benchmarking a polystore, we will test one by one every recommendation and assert if there are relevant in a polystore. This section is threefold. The first objective is to sum up and test each recommendation, the second is to extract recommendations from the test, and the last is to resume each recommendation and the environment or conditions needed to apply the best practice.

4.2.1 Sum up literature and adapt to the polystore environment

The literature review though us some recommendation and best practices. Is it possible to adapt those to the Typhon polystore? We are going to list the evolution allowed and adapt the best practices to the Typhon polystore.

Typhon evolution possibilities

To be able to optimise a database, we need to stay in the possibilities of evolution provided by the Typhon polystore. Those evolution are listed below.

- Add/ rename/ remove attribute
- Add/ rename/ remove entity
- Add/ rename/ remove relation
- Add/drop index
- Add/remove attribute to index
- Add/drop Collection Index
- migrate entity
- merge entities
- Split entity vertical/horizontal
- Enable/Disable relation containment
- Enable/Disable relation opposite
- Change attribute type

To apply those evolution, an user need to formulate change operators. For exemple :

```
changeOperators [
    rename entity Product as NewProduct,
    add attribute title : string [255] to Biography,
    remove relation "User.biography",
    AddIndex {table "Inventory.TabDB" as attributes ("Tag.name")}
    extends tableindex "Inventory.ItemDB" {"Item.shelf"}
]


```

Adaptation to the Typhon polystore

Is the Typhon evolution language expressive enough to let us improve the polystore? We are going to explore all the possibilities.

Add index In each paradigm we have explored, it is recommended to use an index to get query result faster. Indexing will improve read operations for a single or set of queries. Create and update queries can have a loss of performance.[22]

The expressiveness of NoSQL database The Typhon conception does not allow us to increase performance on one specific database. One query's execution time will always be longer on Typhon (with a MariaDB database) than on a MariaDB database, for example. The increase in execution time can be explained by the fact that Typhon is a layer on the database. On the other hand, the time needed to take the output of one query from one database and use it on another database is reduced. Thus we need to take this advantage to intertwine the paradigm. We need to extract the most used ones to model document, graph or key-value database based on queries.
It is recommended to keep the relational database for queries that are not often executed. The document database will be the support for data frequently ask together. The graph database will be the support when a large number of joins are required. Finally, the key-value database will be used
NoSQL database have the advantage to scale

4.2.2 ACID properties

To be able to create a recommendation for a polystore, we need to discuss ACID and BASE properties. The polystore, and more precisely TyphonML, hide the complexity of the database:"Using TyphonML, engineers can model the data that needs to be persistent homogeneously, abstracting over the specificity of the underlying technologies."[42]. If the user can not differentiate in which database is the data, how can we recommend?
A user is choosing by using a relational or NoSQL database. If the user needs scalability, there is more chance that he will select a NoSQL database, and if he need atomicity, he will more likely choose a relational database. In the Typhon environment, the user has a limited view of the model. The border between ACID and BASE is reducing and can lead to surprise for users.

4.3 Implementation

In the previous sections, we explored the literature, list the possibilities of each language and translate them into TyphonML to improve the performance and finally had a discussion about the properties of the data models. The goals of this chapter are to make some recommendation to improve the performance of queries.
To apply the recommendation, we have implemented a model parser to extract info. For example: Do we have an index in a table?
Due to the Typhon development status, which was still in development during my internship and stopped now and let a lot of problems of usability, I was not able to test the improvement of the data models on

actual queries.

4.4 Conclusion

Typhon provides a simple solution to "rules them all". By creating a global solution, unique tuning on a specific database is not easy. For example, the syntax to create a partition key does not exist in Typhon or Typhon lack of user feedback. Performance tuning and recommendation could be highly improved if a solution to get the execution plan as it exists in many languages like with explain keyword in MariaDB. The best way to understand why a query is an anomaly slow is by exploring the execution plan.
Limited by the content available, we have developed a solution to parse a model and a set of queries to make recommendations in the intend to improve performance. This chapter is the first stone of recommendation in polystore. Once the expressiveness of the language will increase or the adoption could generate more and more data, it will become possible to have a more accurate set of rules to improve the polystore.

Chapter 5

Data model visualisation

The literature review though us how to design an interface, create a positive user experience and gave us a methodology with the "what-why-how" model. With all those elements in mind, how can we create a useful visualisation to improve the performance of the polystore? The second chapter mainly focuses on performance assessment, the second focus on how to improve and the goal of this chapter is to help the user to understand and make some good choice while using the polystore. Usage can be multiple, the first is to create the model, the second is to use the model and the third is to make the model evolve. In this chapter, we will try by using the what-why-how methodology to create a useful visualisation. We need to improve the performance by using visualisation techniques (what), to help the user to adopt the solution and have more visibility on the model(why) by creating a UML-like schema that should be integrated into the evolution analytics client (How). This chapter will describe the reflection behind the conception of this tool. The remainder of this chapter is organised as follow. A brief introduction will resume the previous work and how we are going to proceed, then a contribution will explore existing interface and start a thought about the possibilities and implementation of a Typhon model visualisation will be shown.

5.1 Introduction

The Cambridge dictionary definition of visualisation is: "the act or an example of creating an image, etc. to represent something:"[44]. The literature review gave us some insight on how to create an interface, visualisation tools, how to improve the user experiences and even a methodology. All those insights need to be transformed toward the goal to help the user to understand the model and how the polystore works. Starting from the user need and iterate over mockup is the best practice to find a solution that fits the need. Based on the what-why-how model we have defined the objectif of this visualisation: we need to improve the performance by using visualisation techniques (what), to help the user to adopt the solution and have more visibility on the model(why) by creating a UML-like schema that should be integrated into the evolution analytics client (How). In relational database and graph database, the idea to create a visualisation to render a model is common. Respectively, Entity-Relationship diagrams and vertex-edge diagrams, are representations of the real. For key-value and document database, those kinds of visual do not exist because every document is different or it has no sense to create a visual for it.

The UML (Unified Modeling Language) is a standard "helps you specify, visualise, and document models of software systems, including their structure and design, in a way that meets all of these requirements"[43]. There is no standard to represent NoSQL models because they are not fixed in the stone. When we have to define a model in TyphonML, we need to define the structure of each entity for the relational or NoSQL paradigms. This definition gives us a foundation to create a visualisation.

5.2 Contribution

Typhon project has already a high number of interfaces. To create a valuable one, we need to evaluate every interface and understand their purpose. Once every interface have been reviewed, we can start to thing about how we can create an interface to improve the data access performance.

5.2.1 Existing interface

There are multiple ways to interact with the polystore by a graphical user interface or thought web service. There a three graphical user interface. The first one is a plug-in in Eclipse, the second one is the "polystore service UI", and the last is the "evolution analytics client". Each interface has its purpose. We will explore how there are build, find the purpose and the most appropriate place to add a visualisation to help the user create queries.

Eclipse plug-in The first interface is an Eclipse plug-in, figure 5.1. Once the .tml file contains the model, we can inject the model into Typhon. After a couple of steps, we can launch the polystore in a console with the command "docker-compose up –build".

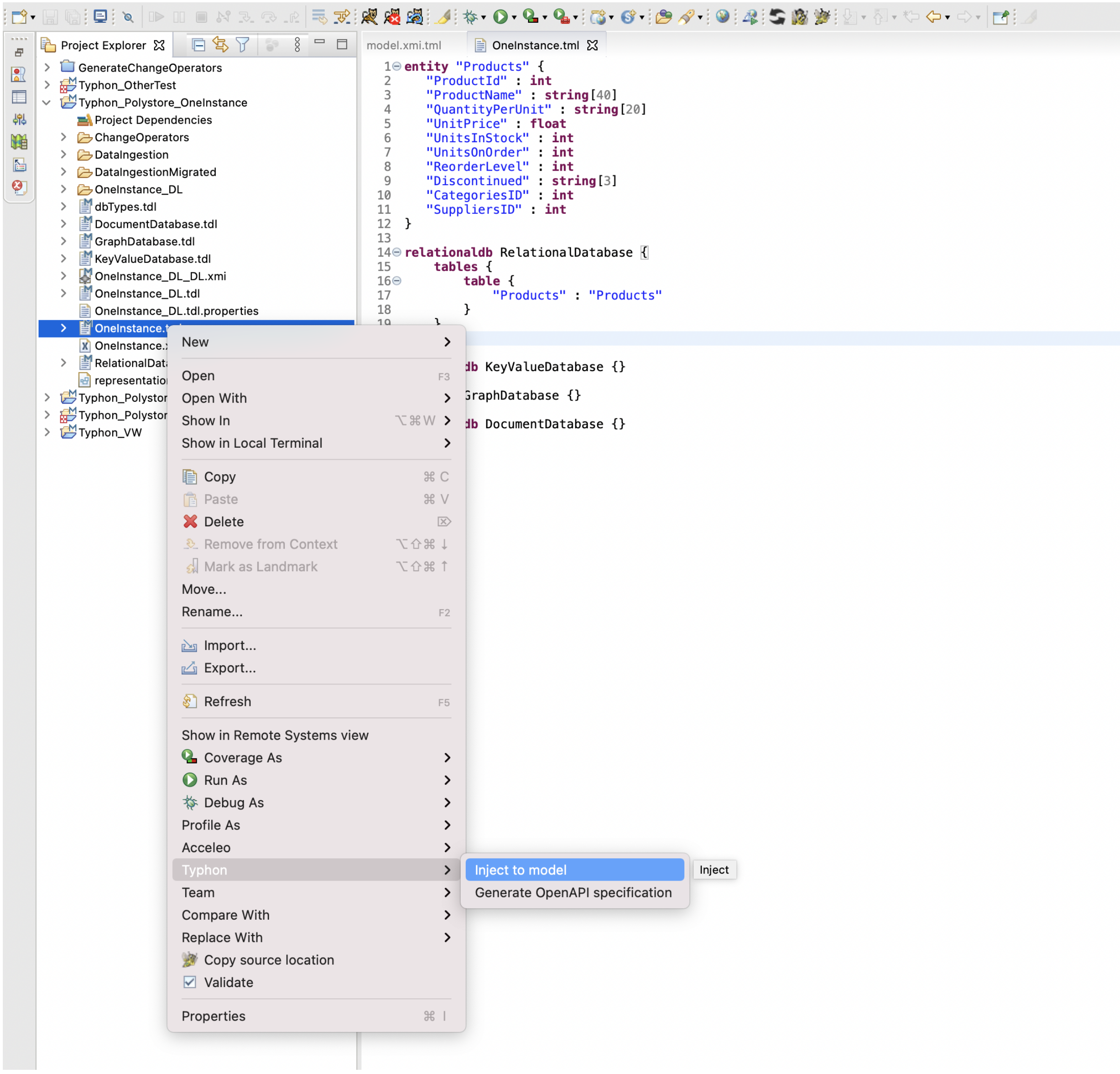

Figure 5.1: Eclipse: Typhon plug-in

This interface is essential to parameters databases, but once the polystore is built, no need to come back to Eclipse.

polystore service UI The second interface is the polystore service UI. This interface helps to manage the polystore. It is composed of multiple panels. The first allow to manage users. The second on to manage database (figure 5.2), the third to manage models (figure 5.3) and the last to perform queries (figure 5.4).

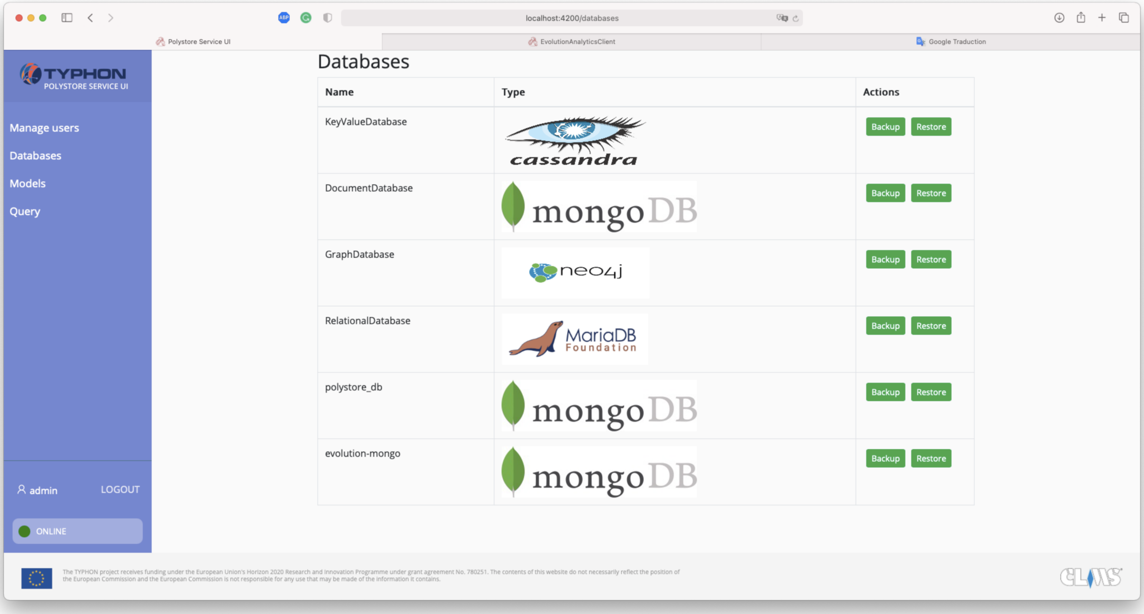

Figure 5.2: Polystore service UI - Databases

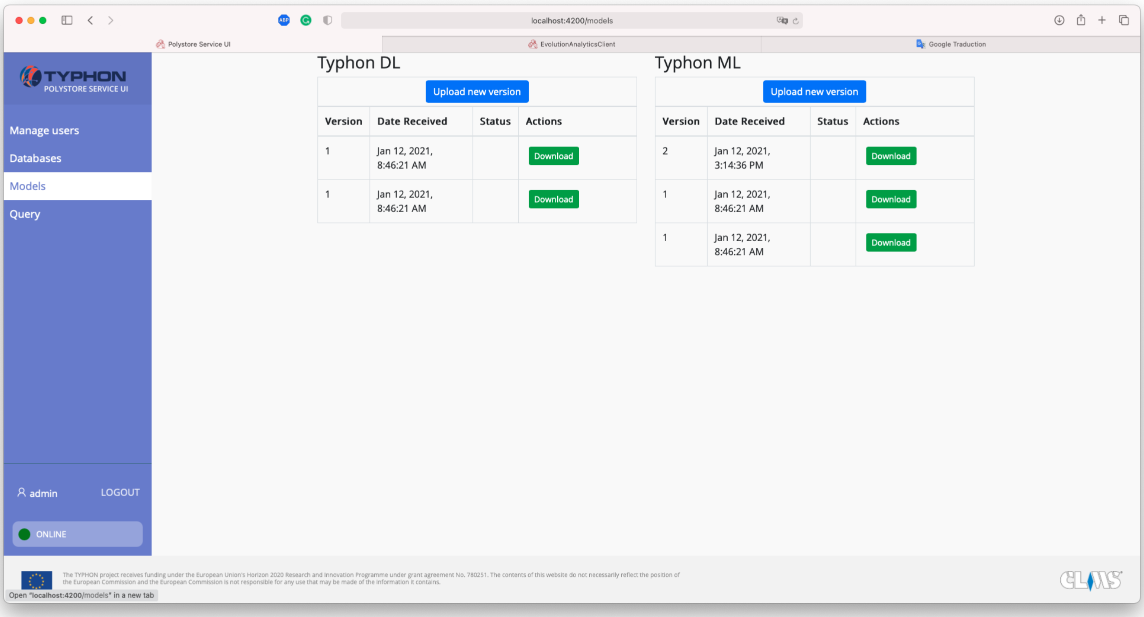

Figure 5.3: Polystore service UI - Models

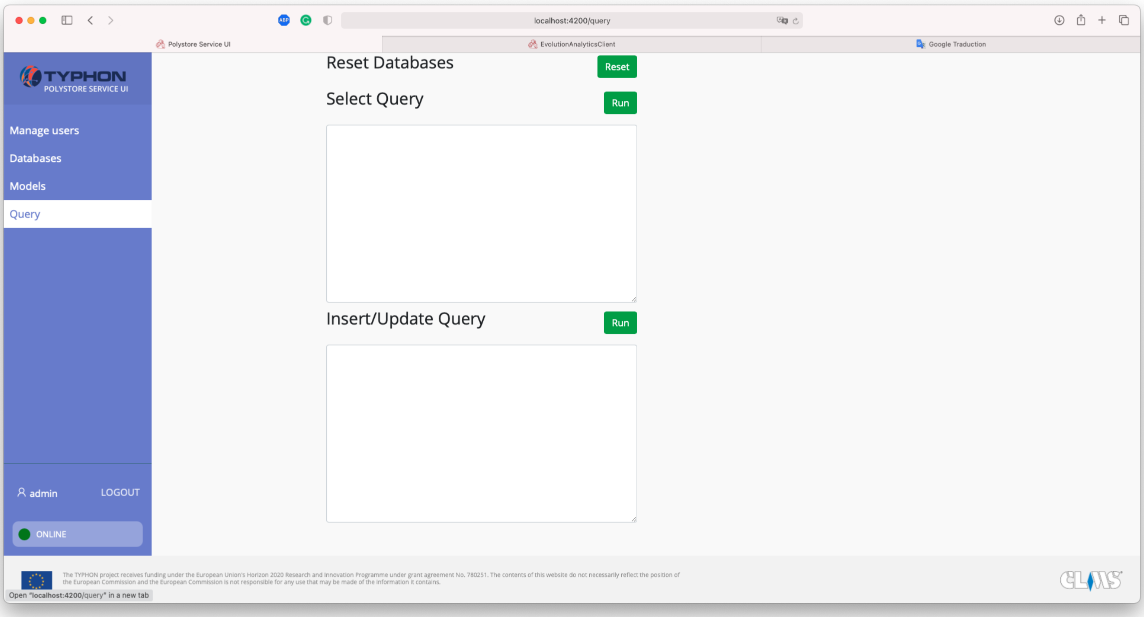

Figure 5.4: Polystore service UI - Query

Evolution analytics client or Typhon Monitoring Interface The third interface is the Evolution analytics client. This interface helps the user to monitor the polystore (figure 5.5), make some change to the data models by applying change operators (figure 5.6) and get statistics on most used category and slowest queries (figure 5.7).

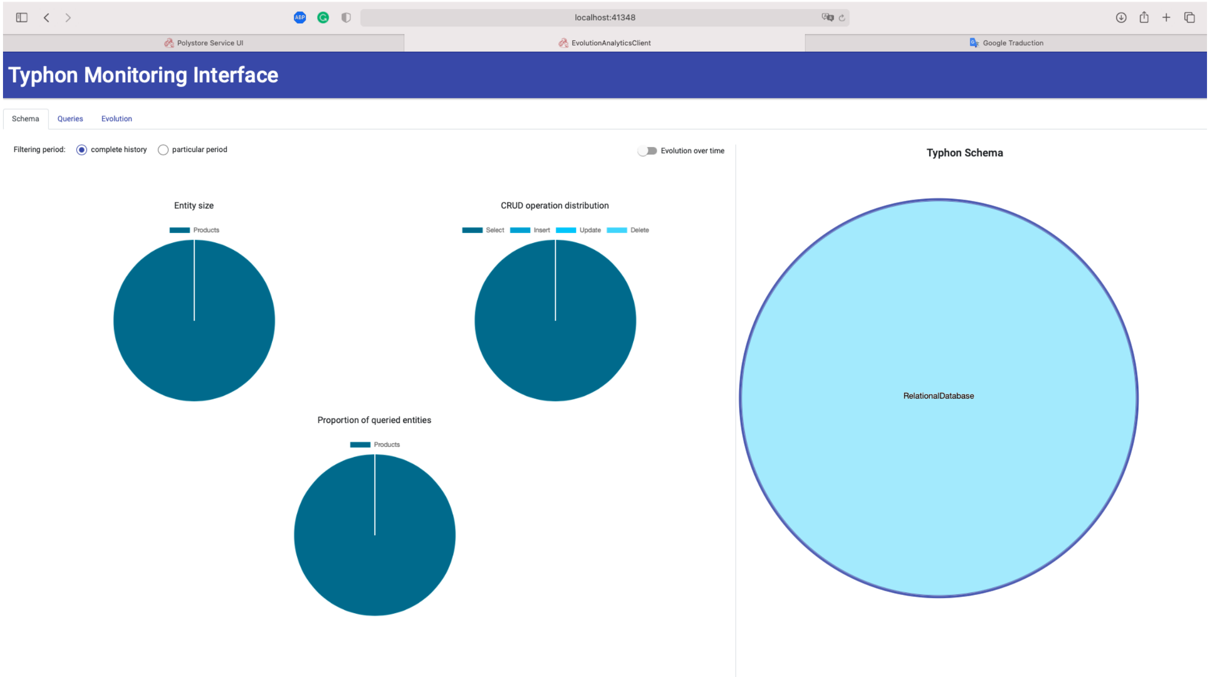

Figure 5.5: Typhon Monitoring Interface - Schema

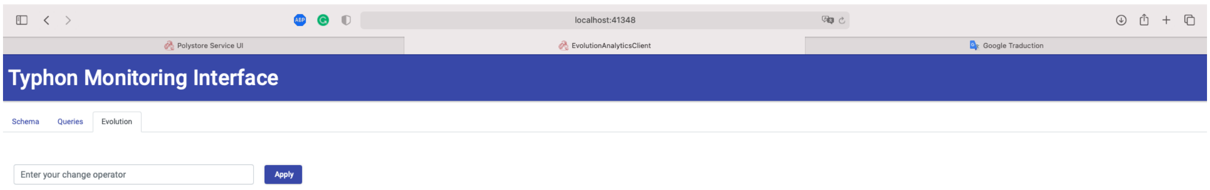

Figure 5.6: Typhon Monitoring Interface - Evolution

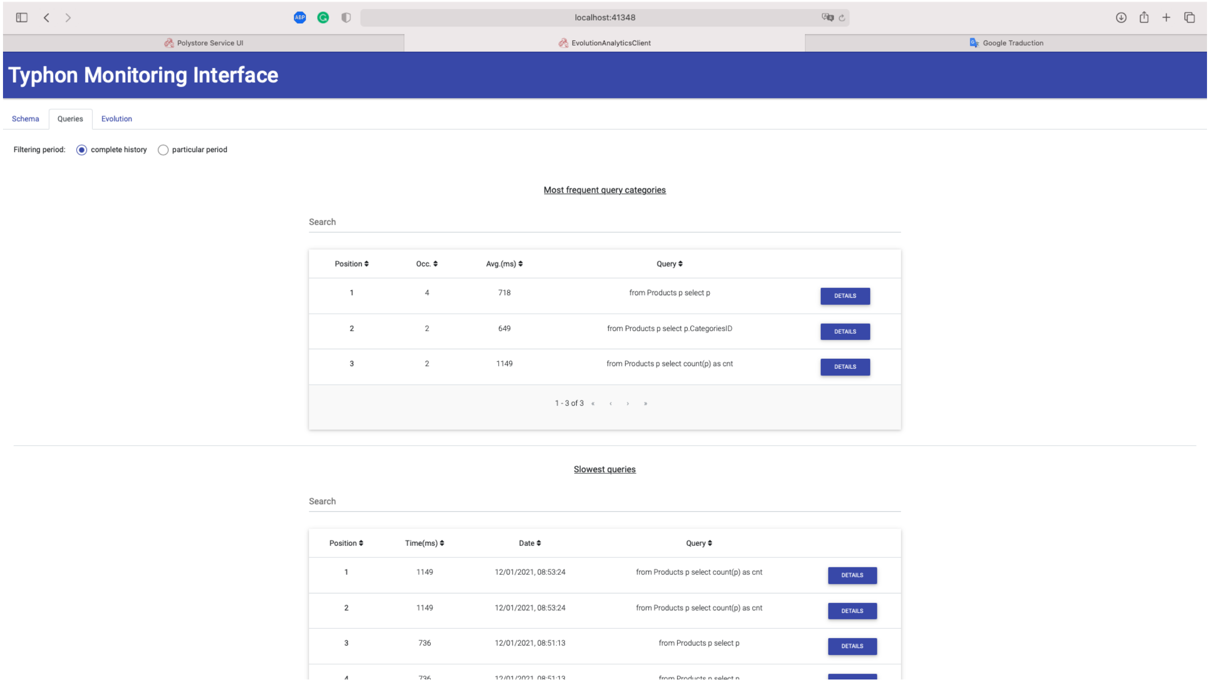

Figure 5.7: Typhon Monitoring Interface - Queries

There are a lot of tools available to manage the polystore. The first tool, a collection of plug-in, is entangled in Eclipse. The second is the Polystore Service UI, and the last is the Evolution Analytics Client. Except in Eclipse, the user can not see the data models. Even when the polystore evolves (Possibilities to change the data models through a web service), it is no longer possible to visualise the model. That leads us to identify a problem: A user can not formulate a correct query without knowing the data models.

5.2.2 Our solution

The purpose of your solution is to improve the understanding and help user to improve data access performance. This interface could be integrated into the Typhon Monitoring Interface because they share a common goal to give the user a better understanding of the database and his queries. The schema into Typhon Monitoring interface has already a visual into the database. On the left pane, three pie charts let us know the size of the entity, number of CRUD operation and the proportion of queried entities and on the right pane, the Typhon schema let us know where we can find each entity. On the other hand with the queries pages, we have two tables providing us with some numbers into the most frequent query pattern and slowest queries. Our approach is to tackle the first problem by exposing to the user a UML-like schema and then on top of that we can add some layers of colours, symbols to represent all the valuable insight that we can get from the analytic tools.
In terms of user experience, the model can quickly became to large for a screen and zooming-out can not be the only solution. We need to create an interface where the user can move in and zoom-in/zoom-out when he need to get an overall idea of the model or to know the composition of each component.

Our solution based on the what-why-how model

What? On short-term : The purpose of our solution is to give back to the user some visibility on the data models and help him improve queries. On long-term : Monitor queries performance by linking query performance and the data models.

Why? We want to help the user to understand what is behind the polystore and create an accurate query.

How? Our solution is to create a UML-like schema to understand what is behind the data models. Our solution will be integrated to the Evolution analytics client or Typhon Monitoring Interface. This choice is due to the wish to monitor and improve the data models.

5.3 Implementation

In the previous section, we have defined the what-why-how of the visualisation. In this section we are going to develop the solution and how the solution works. The model is exposed by Typhon by an API. We need to get the model from this API, transform the model. Once the model is formatted we can then produce a visualisation of the model.

5.3.1 Transform the data models

The first component has for goal to transform the data models from .xml to .json and is programmed in java. This choice has been taken to work with the d3.js library efficiently. After a request to the web service of Typhon to get the data model, it is transformed.

5.3.2 Create a visualisation tools

This implementation is based on a Simple-HTTP-server lunch with "python3 -m http.server". This allows exposing an HTML file to a localhost address. With the well-known d3.js library, the model is drawn.

Those are two examples based on data models from the Typhon project. The first one is from VW (figure 5.8) and the second one is based on the made-up model from Typhon project (figure 5.9).

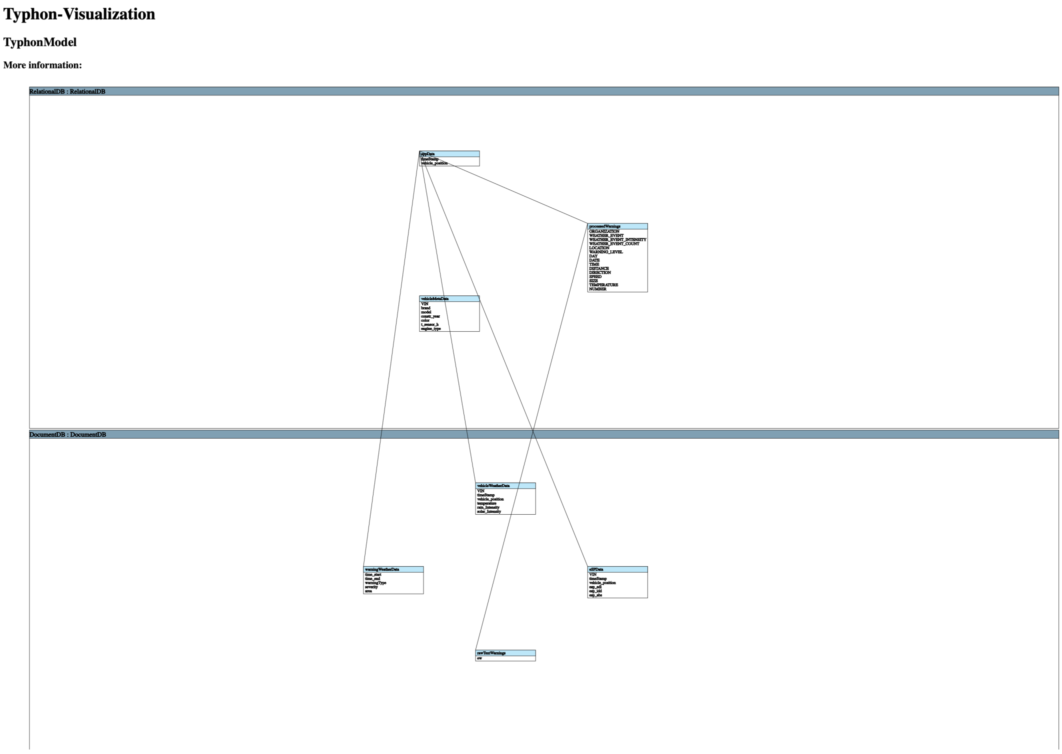

Figure 5.8: Solution with VW Data model

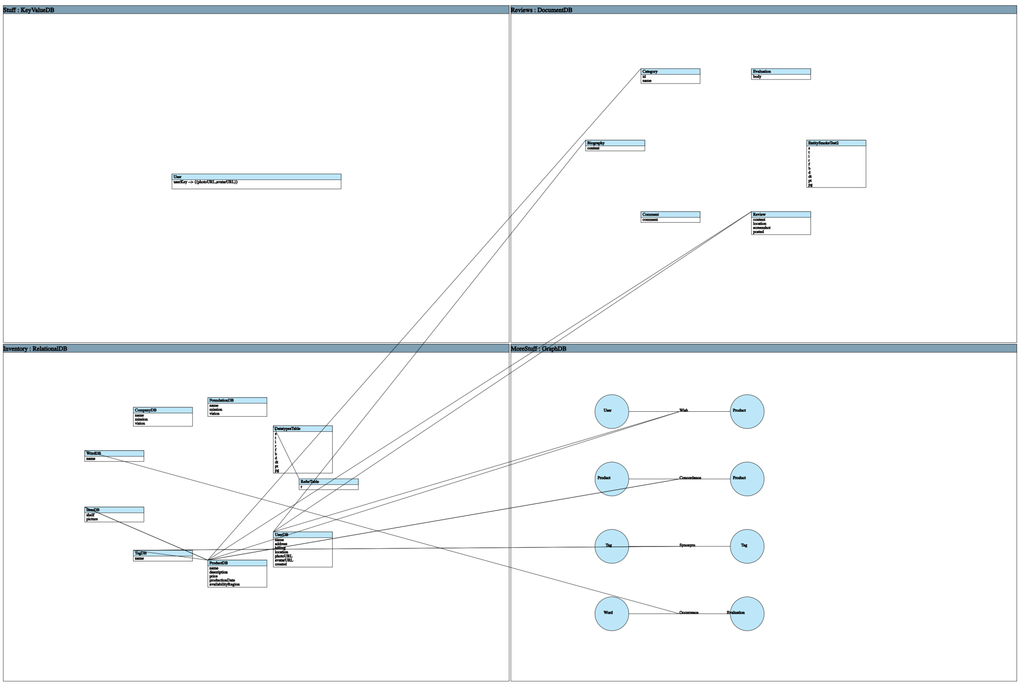

Figure 5.9: Solution with example Data model

5.4 Conclusion

This interface address the most significant problem user can face: missing feedback from the software to understand what is wrong in a query and doing an accurate query based on the underlining paradigm. One problem that this solution has been able to resolve yet is improving the performance of the polystore based on the query performed. One solution is to add layers on top of this solution. Adding a colour map to identifies tables or field which are the most used/ the slowest.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

Performance is one of the main argument toward the adoption of a database management system. By increasing performance, we increase also the attractiveness of the system. A polystore has other advantages than a single paradigm database management system like allowing a user to make queries seamlessly on multiple databases. The polystore architecture add at least one layer on top of relational or NoSQL database (figure 1.1, 1.2), this induce degraded performance compare to one database management system. To avoid the advantage of being able to query multiple database systems seamlessly being wipe out by the disadvantage of poor performance, performance is fundamental.
The first chapter of this thesis gave us some context to understand how hybrid polystore works and divide the work into three sub-topics: benchmarking, recommendation and visualisation. After getting the big picture and context, we had the opportunity to develop a benchmarking tool and test the tool onto Typhon polystore. This benchmarking tool uses the API layer exposed by the Typhon polystore and queries are performed onto a sub-part of the Northwind diagram. Results of the benchmarking were of two types white box and black box result. To be as generic as possible and be paradigm free we choose to ignore white-box result to concentrate on black-box result This implementation was tested on the Typhon polystore but the methodology can be reused on other polystore like BigDawgs. Based on the benchmarking performance, we have extracted some insight into the result and create some recommendation. For example, add an index to a table. Typhon was a living environment because it was still in development. The expressiveness of the language increase, it could be interesting to test new language features. The penultimate chapter was the opportunity to adapt one of the key-feature of a relational model, the UML schema to help the user to understand what was behind the polystore and help him to make accurate queries or change operator. Use a UML schema to represent the NoSQL model is usually not possible due to the flexibility of the language. Typhon reduces this flexibility by requiring the definition of the model in TyphonML. This reduction of the flexibility allows us to define based on this model definition a UML schema. Finding the best way to represent each language was then the next concern. For key-value store and document, we used the same form as for a relational entity that is a box with a name and then some entity linked by relation. For graph database, we use the default representation of a circle for a node and a relation between nodes represented by a line.

6.2 Future Works

My experience in Computer sciences and in the Typhon project has tough me meaningful insight. In this section, I am going to explore what's could be done to integrate my solution and what could done to improve the Typhon polystore. Each part of this work could be integrated into the project at different place. We are going to describe the integration of each part.

6.2.1 Benchmark integration

The benchmarking tools could help to know what is happening behind the door and what could cause some latency.
To be accurate this benchmark should reduce from first of all the user's query and based on the data of the users. Create a scheduled jobs based on the n most used query and making a report for the user could help him to improve the database structure.
The benchmarking could also be adapted to be run on multiple systems like BigDawg, another implementation of a polystore.

6.2.2 Recommendation integration

Recommendation can be static and always be a good idea like fitting the type of a field to the data or changing based on the model and the query like adding an index. What could improve a model on one specific use case could cause some latency on another and is a key problem to keep in mind.
Recommendation should be linked to the benchmark to fit to the users need but to make it possible a step need to be overcome by Typhon to make the database behind the polystore invisible. The polystore should be able to direct a query toward a paradigm or another based on the query. For example a query with a lot of joins could be more efficient into a graph database. What query, when to use one or the other is still to discover once the seamless experience will be effective.

6.2.3 Visualisation integration

For the visualisation, my idea was clear from the beginning create a representation of the data model and add a heat map on top of it to see where are performed most used query, where are performed less efficiency queries, etc. With the deadline approaching, I had to reduce the starting ambition. The lack of data and the complexity to connect the model, queries performance and model visualisation forced me to reduce my ambition. I would be glad if someone could improve the visualisation by adding this feature and test it at larger scale.
The visualisation is composed of two part. A back-end responsible to get the data model, transform it and expose it to the front-end. The second part is a front-end, which with the help of d3.js give form to the model. As the front-end is uncoupled from Typhon, it could be reused with other polystore.

6.2.4 User empowerment

We leave in an world were customers take more and more power. This customers empowerment can be related to his capabilities to quickly change, compare, test, comment and share. The extract the polystore from the abyss and make it popular the user feedback is critical.
The idea behind the polystore start from a customer need, to be able to make query on multiple database, by his development the user was more put on a side and to be able to make some query on the polystore became complicated due to the fact that the user has no view on the model or on what is happening with his query.

6.2.5 ACID and BASE applied to polystore

ACID and BASE are respectively properties of relational, NoSQL database. As polystore have for purpose to hide underling database from the user. A thinking need to be orchestrate around the problematic of ACID and BASE properties. Should we keep the most restricted properties, ACID, or should we take the BASE as root properties? Is it possible to make them coexist inside a polystore and keep ACID properties for relational database and BASE for NoSQL and get other properties for the polystore?

Appendix A

Glossary

Abbreviation - Word	**Signification**
DBMS	Database management system
TyphonML	Typhon Modelling Language
TyphonQL	Typhon Query Language

Appendix B

Configuration

B.1 Hardware and software configuration

B.1.1 Computer

- Configuration
  - Model Name: MacBook Pro
  - Model Identifier: MacBookPro16,1
  - Processor Name: 6-Core Intel Core i7
  - Processor Speed: 2,6 GHz
  - Number of Processors: 1
  - Total Number of Cores: 6
  - L2 Cache (per Core): 256 KB
  - L3 Cache: 12 MB
  - Hyper-Threading Technology: Enabled
  - Memory: 16 GB (2*8GB - DDR4 - 2667 Mhz)
  - Boot ROM Version: 1037.147.4.0.0 (iBridge: 17.16.16610.0.0,0)
  - SSD Memory (Capacity: 500,28 GB (500.277.792.768 bytes), Model: APPLE SSD AP0512N, Link Width: x4,Link Speed: 8.0 GT/s)
- System Software:
  - System Version: macOS 10.15.7 (19H2)
  - Kernel Version: Darwin 19.6.0

B.1.2 Docker

- Version :
  - v19.03.13
- Ressources :
  - CPUs : 6
  - Memory: 8.00GB
  - Swap : 1GB
  - Disk image : 59.6GB

B.1.3 Typhon

Version

- neo4j (Images ID: ce22583052bf)
- mongo (Images ID: c97feb3412a3)
- swatengineering/typhonql-server (Images ID: 64af3c4917e4)
- clms/typhon-polystore-api (Images ID: 14fd01b9a1b5)
- universityofyork/typhon-analytics (Images ID: 47534925b54e)
- meuriceloup/typhon-evolution-analytics-backend (Images ID: f8c6b9ddf1ed)
- meuriceloup/typhon-evolution-analytics-client (Images ID: e32838c674cb)
- meuriceloup/typhon-evolution-analytics-java (Images ID: bf78e8aaa472)
- cassandra (Images ID: 8baadf8d390f)
- mariadb (Images ID: 3a348a04a815)
- clms/typhon-polystore-ui (Images ID: 14fd01b9a1b5)
- zolotas4/typhon-analytics-auth-all (Images ID: 90904fd7a2dc)
- wurstmeister/kafka (Images ID: 9a5842c217a8)
- wurstmeister/zookeeper (Images ID: 3f43f72cb283)

Database Image Configuration

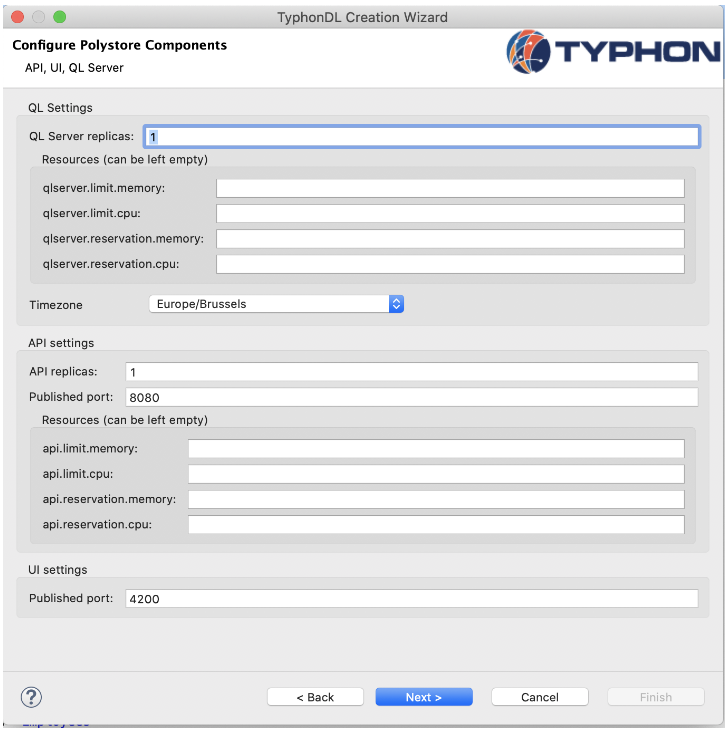

Figure B.1: Configure Polystore Components

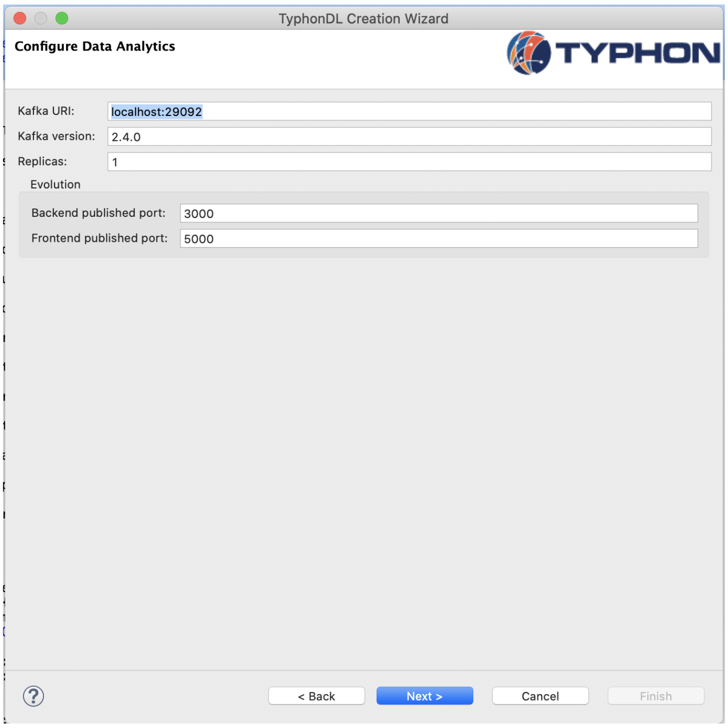

Figure B.2: Configure Data Analytics

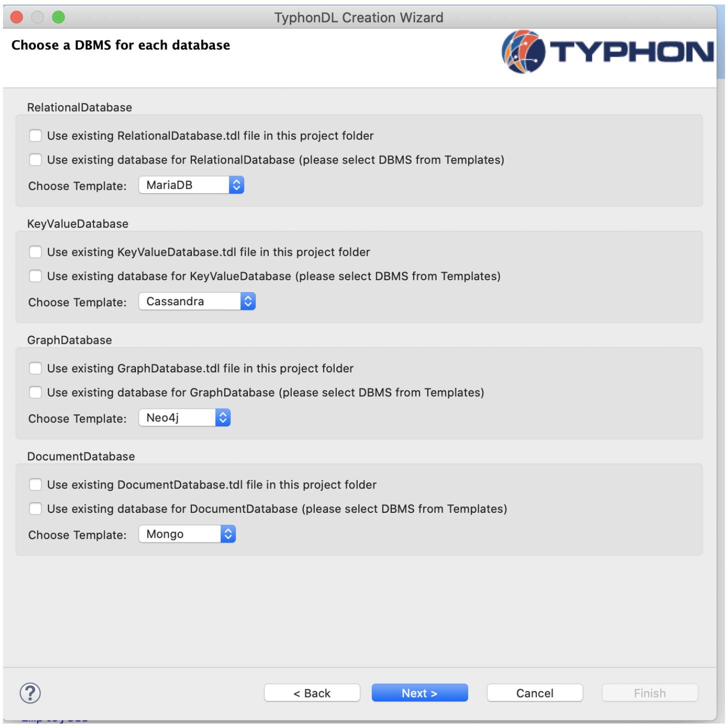

Figure B.3: Choose a DBMS for each database

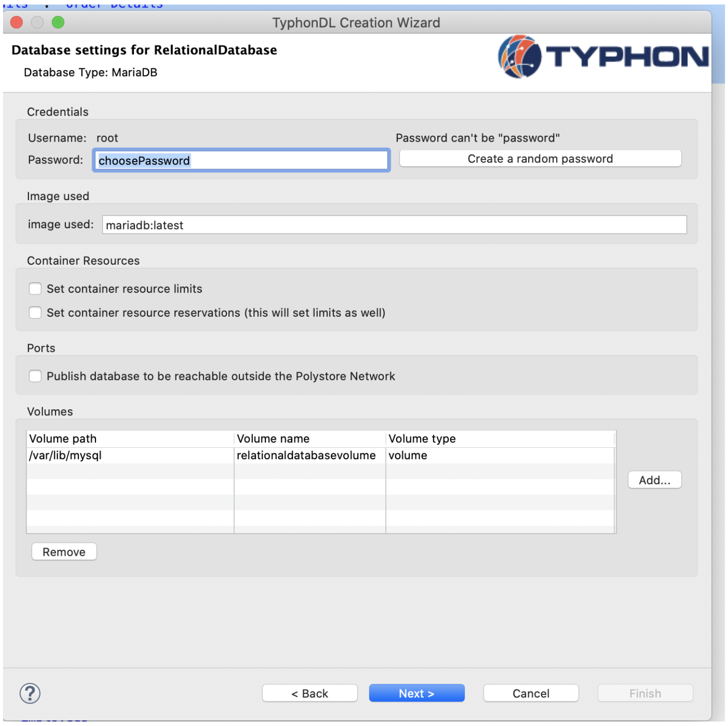

Figure B.4: Database settings for RelationalDatabase

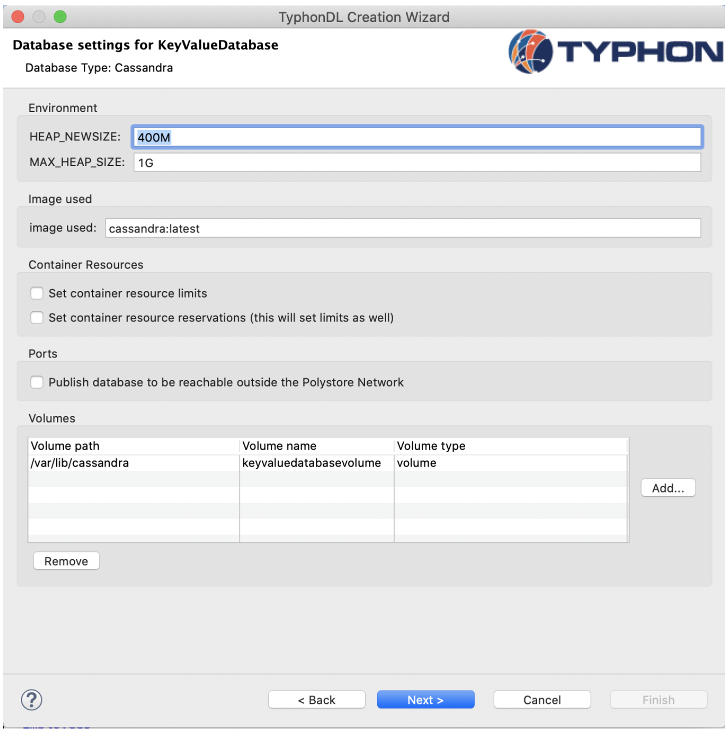

Figure B.5: Database settings for KeyValueDatabase

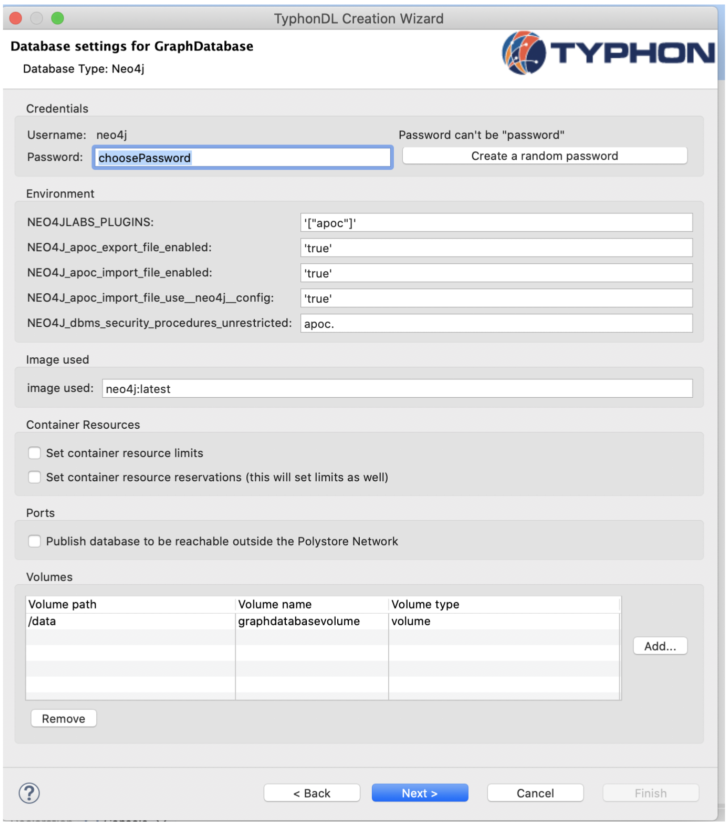

Figure B.6: Database settings for GraphDatabase

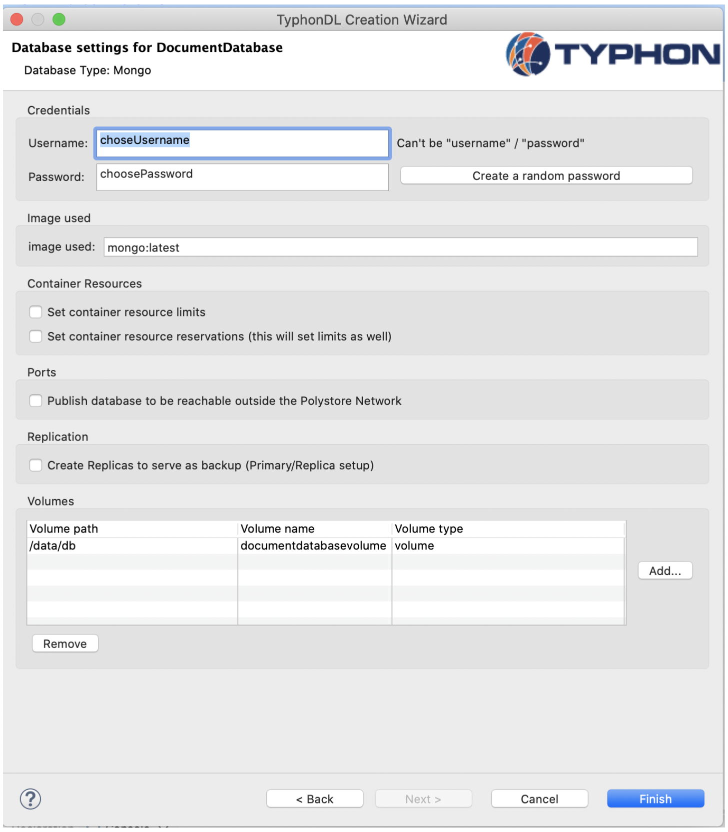

Figure B.7: Database settings for DocumentDatabase

Appendix C

Data Ingestion Tool

The following procedure to extract model and data from an existing database is the MacOS version. For an extensive description please refer to documentation available at `https://github.com/typhon-project/typhon-evolution` or in the Typhon User Guide.

C.1 Data ingestion tool

1. Download source code from
2. Build source code

```
$ cd data_ingestion
$ mvn clean install

```

3. Modifies the file 'extract.properties' available in the generate file under the data_ingestion/target directory.

```
URL=jdbc:mysql://relational.fit.cvut.cz:3306/northwind
DRIVER=org.mariadb.jdbc.Driver
USER=guest
PASSWORD=relational
SCHEMA=northwind

```

4. Execute the extraction with (Don't forget to create a directory output or to specify a valid output) On linux :

```
$ bash data_ingestion.sh -extract extract.properties output

```

5. Follow Typhon user guide to create a polystore and import shema.tml into your project.
6. Launch the polystore
7. The last step consists in executing the data ingestion scripts (.tql) generated at Step 1.

Appendix D

Queries

D.1 Typhon Query Language (TyphonQL)

This chapter concentrate all the queries used during the benchmarking process. It's organised by type of queries (CRUD). Every query belong to at least one set of query defined in 3.1
The syntax is defined using this rules:

- Keyword are bold
- Text to replace depending on the context are in italics.
- Text between parentheses are optional
- "*" signify that the text between parentheses just before the symbol could be present 0 or multiple times.
- The symbol pipe ("—") signify multiple possibility like "or"
- TableName: the name of the table or entity where to search.
- Alias: a letter or multiple letter which relate to an table name.
- Attributes : attributes from a table or entity.
- Condition: a filter to satisfied on an entity.

D.1.1 Create Queries

Syntax

```
insert TableName {attributes : value (, attributes: {value)* }
```

D.1.2 Read Queries

Syntax

```
from TableName Alias(,TableName Alias)*
select Alias(.Attribute),(Alias(.Attribute))*
where Condition
```

Syntax Condition

```
Alias.Attribute Relation Variable|Alias(.Attribute)
(,Alias.Attribute Relation Variable|Alias(.Attribute))*
```

D.1.3 Syntax Relation

like in && ——

D.1.4 Update Queries

Syntax

```
 update TableName Alias
 where condition
 set {attribute:value(,attibute:value)*}
```

D.1.5 Delete Queries

Syntax

```
delete TableName Alias
where condition

```

D.2 Queries

D.2.1 Heavy read

```
////// HeavyRead50 Create7 Read35 Update5 Delete3

from Employees e
select e.LastName, e.FirstName

from Customers c
select c.ContactName

from Shippers s
select s.CompanyName, s.Phone

from Customers c
select c.CompanyName
where c.Country == "UK"
```

```

from Orders o
select o.Freight
where o.Freight >= 100

from Suppliers s
select s.@id

from Products p
select p.@id
where p.UnitPrice > 50 && p.UnitsInStock >= 20

from Customers c
select c.@id
where c.Country=="Spain" && c.City=="Madrid"

from Order_Details o
select o.@id, o.Quantity, o.UnitPrice
where o.Quantity > 10

from Products p
select p.ProductName, p.UnitPrice
where p.ProductName like "'s"

from Customers c
select c.ContactName, c.CompanyName, c.Phone, c.Fax
where c.ContactTitle == "Owner"

from Product p
select p.ProductName, p.UnitPrice, p.UnitsInStock, p.UnitsOnOrder, p.Discontinued

from Categories c
select c.CategoryName, c.Description, c.Picture

from Products p, Suppliers s
select p.ProductName, s.CompanyName
where p.Suppliers == s

from Orders o
select o.Order_Details

from Customers c, Orders o
select c.CompanyName, o.OrderDate, o.Customers
where c.Orders == o && c.Country!="USA"

from Orders o, Order_Details od
select o.@id, od.Quantity, od.Products
where o.Order_Details == od

from Suppliers s
select s.Products

from Customers c
select c.Orders

from Categories c
select c.Products
```

```

from Employees e
select e.Employees_1
where e.Country=="USA"

from Region r
select r.Territories
where r.RegionDescription=="Eastern"

from Orders o
select o.Customers

from Orders o
select o.Employees
where o.OrderDate like "1996"

from Customers c, Orders o, Employees e
select c.CompanyName, e.LastName, e.FirstName
where c==o.Customers && o==e.Orders

from Employee e
select e.@id, e.Birthdate
where e.Birthdate like "-01-"

from Employee e
select e.@id, e.LastName
where e.HireDate like "1992"

from Orders o, Shippers s
select o.OrderDate, o.@id, o.OrderDate, o.RequiredDate, s.CompanyName, s.Phone
where o== s.Orders && s.CompanyName == "Speedy Express"

from Products p
select p.ProductName
where p.ProductName like "ch"

from Customers c, Orders o, Order_Details od, Products p
select c.CompanyName, p.ProductName, od.Quantity
where c.CompanyName == "White Clover Markets" && c== o.Customers && o == od.
    Orders && od== p.Order_Details

from Employees e
select e.TitleOfCourtesy

from Products p
select p.ProductName, p.UnitPrice, p.UnitsInStock

from Products p
select p.ProductName, p.UnitPrice, p.UnitsInStock
where p.Discontinued=="0"

from Products p
select p.ProductName, p.UnitPrice, p.UnitsInStock
where p.Discontinued=="1"

from Suppliers s
select s.Phone
```

```
where s.CompanyName=="Exotic Liquids"

insert Suppliers {
 @id:#GenHRComp,
 CompanyName : "gen Company",
 ContactName : "gen and son",
 ContactTitle : "gen Georges",
 Address : "Angel's street ,14",
 City : "Namur",
 PostalCode : "5000",
 Country : "Belgium"
}

insert Categories {
  @id:#GenHRCat,
  CategoryName : "gen Category",
  Description : "This is a gen Category"
}

update Products p
where p.ProductName =="Chai"
set {price :+ 5}

insert Products {
  @id:#GenHRProduct,
  ProductName : "gen Product",
  QuantityPerUnit : "One unit",
  UnitPrice : 1.0,
  UnitsInStock : 10,
  UnitsOnOrder : 0,
  ReorderLevel : 1,
  Discontinued : "0",
  Categories: #GenHRCat,
  Suppliers: #GenHRComp
}

insert Products {
  @id:#GenHRProduct2,
  ProductName : "gen Product two",
  QuantityPerUnit : "Two unit",
  UnitPrice : 5.0,
  UnitsInStock : 2,
  UnitsOnOrder : 10,
  ReorderLevel : 2,
  Discontinued : "0",
  Categories: #GenHRCat,
  Suppliers: #GenHRComp
}

update Suppliers s
where s.@id == #GenHRComp
set {Products +: [#GenHRProduct,#GenHRProduct2]}

update Categories c
where c.@id == #GenHRCat
set {Products +: [#GenHRProduct,#GenHRProduct2]}
```

```
insert Employees {
  @id:#GenHREmployee,
  LastName : "gen",
  FirstName : "Bob",
  Title : "Mr",
  BirthDate : "2000-01-01",
  HireDate : "2020-01-01",
  Notes : "Not available",
  PhotoPath : "http://google.com",
  Salary : 1000.0
}

insert Customers {
  @id:#GenHRCons,
  CustomerID : "GENCU",
  CompanyName : "Gen Customer",
  ContactName : "Ana Lyse"
}

insert Orders {
  @id : #GenHROrder,
  OrderDate : "2020-11-01",
  Employee : #GenHREmployee,
  Customer : #GenHRCons
}

update Employees e
where e.@id == #GenHREmployee
set {Orders +: [#GenHROrder]}

update Customers c
where c.@id == #GenHRCons
set {Orders +: [#GenHROrder]}

delete Orders o
where o.@id == #GenHROrder

delete Customers c
where c.@id == #GenHRCons

delete Employees e
where e.@id == #GenHREmployee
```

Appendix/Queries/HeavyRead.tql

D.2.2 Heavy create and update

```
////// Heavycreateandupdate50 Read3 Create22 Update22 Delete3

from Orders o, Shippers s
select o.OrderDate, o.@id, o.OrderDate, o.RequiredDate, s.CompanyName, s.Phone
where o== s.Orders && s.CompanyName == "Speedy Express"

from Products p
select p.ProductName, p.@id, p.UnitPrice, p.UnitsInStock, p.UnitsOnOrder
where p.ProductName like "ch"
```

```

from Customers c, Orders o, Order_Details od, Products p
select c.CompanyName, p.ProductName, od.Quantity
where c.CompanyName == "White Clover Markets" && c== o.Customers && o == od.
    Orders && od== p.Order_Details

insert Suppliers {
 @id:#GenHCComp,
 CompanyName : "generate Company",
 ContactName : "generate and son",
 ContactTitle : "generate Georges",
 Address : "Angel's street , 14",
 City : "Namur",
 PostalCode : "5000",
 Country : "Belgium"
}

insert Categories {
  @id:#GenHCCat,
  CategoryName : "generate Cat",
  Description : "This is a generate Category"
}

update Products p
where p.ProductName =="Chai"
set {price :+ 5}

insert Products {
  @id:#GenHCProduct,
  ProductName : "generate Product",
  QuantityPerUnit : "One unit",
  UnitPrice : 1.0,
  UnitsInStock : 10,
  UnitsOnOrder : 0,
  ReorderLevel : 1,
  Discontinued : "0",
  Categories: #GenHCCat,
  Suppliers: #GenHCComp
}

insert Products {
  @id:#GenHCProduct2,
  ProductName : "generate Product two",
  QuantityPerUnit : "Two unit",
  UnitPrice : 5.0,
  UnitsInStock : 2,
  UnitsOnOrder : 10,
  ReorderLevel : 2,
  Discontinued : "0",
  Categories: #GenHCCat,
  Suppliers: #GenHCComp
}

insert Products {
  @id:#GenHCProduct3,
  ProductName : "generate Product three",
  QuantityPerUnit : "One package",
```

```
  UnitPrice : 45.0,
  UnitsInStock : 3,
  UnitsOnOrder : 4,
  ReorderLevel : 1,
  Discontinued : "0",
  Categories: #GenHCCat,
  Suppliers: #GenHCComp
}

update Suppliers s
where s.@id == #GenHCComp
set {Products +: [#GenHCProduct,#GenHCProduct2,#GenHCProduct3]}

update Categories c
where c.@id == #GenHCCat
set {Products +: [#GenHCProduct,#GenHCProduct2,#GenHCProduct3]}

insert Employees {
  @id:#GenHCEmployee,
  LastName : "Generate",
  FirstName : "Bob",
  Title : "Mr",
  BirthDate : "2000-01-01",
  HireDate : "2020-01-01",
  Notes : "Not available",
  PhotoPath : "http://google.com",
  Salary : 1000.0
}

insert Customers {
  @id:#GenHCCons,
  CustomerID : "GENCU",
  CompanyName : "Gen Customer",
  ContactName : "Ana Lyse"
}

insert Orders {
  @id:#GenHCOrder,
  OrderDate : "2020-11-01",
  Employee : #GenHCEmployee,
  Customer : #GenHCCons
}

insert Order_Details {
  @id: #OrderDetails,
  UnitPrice : 1.0,
  Quantity : 5,
  Discout : 0.0,
  Products : #GenHCProduct,
  Orders : #GenHCOrder
}

insert Order_Details {
  @id: #OrderDetails2,
  UnitPrice : 5.0,
  Quantity : 2,
  Discout : 0.5,
```

```
  Products : #GenHCProduct2,
  Orders : #GenHCOrder
}

update Orders o
where o.@id == #GenHCOrder
set {Order_Details +: [#OrderDetails2,#OrderDetails]}

update Products p
where p.@id == #GenHCProduct
set {Order_Details +: [#OrderDetails]}

update Products p
where p.@id == #GenHCProduct2
set {Order_Details +: [#OrderDetails2]}

update Employees e
where e.@id == #GenHCEmployee
set {Orders +: [#GenHCOrder]}

update Customers c
where c.@id == #GenHCCons
set {Orders +: [#GenHCOrder]}

insert Employees {
  @id:#GenHCEmployee2,
  LastName : "Generate",
  FirstName : "Mario",
  Title : "Mr",
  BirthDate : "1999-01-01",
  HireDate : "2020-06-01",
  Notes : "Not available",
  PhotoPath : "http://facebook.com/Mario",
  Salary : 2000.0,
  Employees_1 : #GenHCEmployee
}

update Employees e
where e.@id == #GenHCEmployee
set {Employees +: [#GenHCEmployee2]}

insert Orders {
  @id: #GenHCOrder2,
  OrderDate : "2020-11-01",
  Employees : #GenHCEmployee,
  Customers : #GenHCCons
}

insert Order_Details{
  @id: #OrderDetails3,
  UnitPrice : 1.0,
  Quantity : 5,
  Discout : 0.0,
  Products : #GenHCProduct,
  Orders : #GenHCOrder
}

```

```
insert Order_Details{
  @id: #OrderDetails4 ,
  UnitPrice : 5.0 ,
  Quantity : 2 ,
  Discout : 0.5 ,
  Products : #GenHCProduct2 ,
  Orders : #GenHCOrder
}

insert Order_Details{
  @id: #OrderDetails5 ,
  UnitPrice : 2.0 ,
  Quantity : 1 ,
  Discout : 1 ,
  Products : #GenHCProduct3 ,
  Orders : #GenHCOrder
}

update Orders o
where o.@id == #GenHCOrder2
set {Order_Details +: [#OrderDetails3 ,#OrderDetails4 ,#OrderDetails5]}

update Products p
where p.@id == #GenHCProduct
set {Order_Details +: [#OrderDetails3]}

update Products p
where p.@id == #GenHCProduct2
set {Order_Details +: [#OrderDetails4]}

update Employees e
where e.@id == #GenHCEmployee2
set {Orders +: [#GenHCOrder2]}

update Customers c
where c.@id == #GenHCCons
set {Orders +: [#GenHCOrder2]}

insert Customers {
  @id:#GenHCCons2 ,
  CustomerID : "GENTW" ,
  CompanyName : "Twin Customer" ,
  ContactName : "Ty Phon"
}

insert Orders {
  @id: #GenHCOrder3 ,
  OrderDate : "2020-11-01" ,
  Employees : #GenHCEmployee2 ,
  Customers : #GenHCCons2
}

update Customers c
where c.@id == #GenHCCons2
set {Orders +: [#GenHCOrder3]}

update Employees e
```

```
where e.@id == #GenHCEmployee2
set {Orders +: [#GenHCOrder3]}

insert Order_Details{
  @id: #OrderDetails6,
  UnitPrice : 2.0,
  Quantity : 1,
  Discout : 1,
  Products : #GenHCProduct,
  Orders : #GenHCOrder3
}

update Products p
where p.@id == #GenHCProduct
set {Order_Details +: [#OrderDetails6]}

insert Order_Details{
  @id: #OrderDetails7,
  UnitPrice : 2.0,
  Quantity : 1,
  Discout : 1,
  Products : #GenHCProduct3,
  Orders : #GenHCOrder3
}

update Products p
where p.@id == #GenHCProduct3
set {Order_Details +: [#OrderDetails7]}

insert Order_Details{
  @id: #OrderDetails8,
  UnitPrice : 2.0,
  Quantity : 1,
  Discount : 1,
  Products : #GenHCProduct2,
  Orders : #GenHCOrder3
}

update Products p
where p.@id == #GenHCProduct2
set {Order_Details +: [#OrderDetails8]}

insert Categories {
  @id:#GenHCCat2,
  CategoryName : "generate Cat n2",
  Description : "This is a generate Category number two"
}

insert Products {
  @id:#GenHCProduct4,
  ProductName : "Generate Product four",
  QuantityPerUnit : "One pounds",
  UnitPrice : 10000.0,
  UnitsInStock : 1,
  UnitsOnOrder : 0,
  ReorderLevel : 1,
  Discontinued : "0",
```

```
  Categories: #GenHCCat2,
  Suppliers: #GenHCComp
}

update Categories c
where c.@id == #GenHCCat2
set {Products +: [#GenHCProduct4]}

update Order_Details o
where o.@id == #OrderDetails8
set {UnitPrice : 1000, Discount : 0}

delete Orders o
where o.@id == #GenHCOrder

update Products p
where p.ProductName =="Chai"
set {price :- 5}

delete Customers c
where c.@id == #GenHCCons

delete Employees e
where e.@id == #GenHCEmployee
```

Appendix/Queries/HeavyCreate.tql

D.2.3 Balanced read, write and update

```
////// Balanced50 Create15 Read15 Update15 delete5

from Customers c
select c.ContactName

from Employees e
select e.LastName, e.FirstName

from Shippers s
select s.CompanyName, s.Phone

from Customers c
select c.CompanyName
where c.Country == "UK"

from Orders o
select o.Freight, o.@id, o.OrderDate
where o.Freight >= 100

from Suppliers s
select s.CompanyName

from Products p
select p.ProductName
where p.UnitPrice > 50 && p.UnitsInStock >= 20

from Customers c
```

```
select c.CustomerID, c.ContactName, c.CompanyName
where c.Country=="Spain" && c.City=="Madrid"

from Order_Details o
select o.@id, o.Quantity, o.UnitPrice
where o.Quantity > 10

from Products p
select p.ProductName, p.UnitPrice
where p.ProductName like "'s"

from Customers c
select c.ContactName, c.CompanyName, c.Phone, c.Fax
where c.ContactTitle == "Owner"

from Product p
select p.ProductName, p.UnitPrice, p.UnitsInStock, p.UnitsOnOrder, p.Discontinued

from Categories c
select c.CategoryName, c.Description, c.Picture

from Products p, Suppliers s
select p.ProductName, s.CompanyName
where p == s.Products

from Customers c, Orders o, Order_Details od, Products p
select c.CompanyName, p.ProductName, od.Quantity
where c.CompanyName == "White Clover Markets" && c== o.Customers && o == od.
    Orders && od== p.Order_Details

insert Suppliers {
 @id:#GenBComp,
 CompanyName : "Generate Company",
 ContactName : "Generate and son",
 ContactTitle : "Generate Georges",
 Address : "Angel's street, 14",
 City : "Namur",
 PostalCode : "5000",
 Country : "Belgium"
}

insert Categories {
  @id:#GenBCat,
  CategoryName : "Generate Cat",
  Description : "This is a generate Category"
}

update Products p
where p.ProductName =="Chai"
set {price :+ 5}

insert Products {
  @id:#GenBProduct,
  ProductName : "Generate Product",
  QuantityPerUnit : "One unit",
  UnitPrice : 1.0,
  UnitsInStock : 10,
```

```
  UnitsOnOrder : 0,
  ReorderLevel : 1,
  Discontinued : "0",
  Categories: #GenBCat,
  Suppliers: #GenBComp
}

insert Products {
  @id:#GenBProduct2,
  ProductName : "Generate Product two",
  QuantityPerUnit : "Two unit",
  UnitPrice : 5.0,
  UnitsInStock : 2,
  UnitsOnOrder : 10,
  ReorderLevel : 2,
  Discontinued : "0",
  Categories: #GenBCat,
  Suppliers: #GenBComp
}

insert Products {
  @id:#GenBProduct3,
  ProductName : "Generate Product three",
  QuantityPerUnit : "One package",
  UnitPrice : 45.0,
  UnitsInStock : 3,
  UnitsOnOrder : 4,
  ReorderLevel : 1,
  Discontinued : "0",
  Categories: #GenBCat,
  Suppliers: #GenBComp
}

update Suppliers s
where s.@id == #GenBComp
set {Products +: [#GenBProduct,#GenBProduct2,#GenBProduct3]}

update Categories c
where c.@id == #GenBCat
set {Products +: [#GenBProduct,#GenBProduct2,#GenBProduct3]}

insert Employees {
  @id:#GenBEmployee,
  LastName : "Generate",
  FirstName : "Bob",
  Title : "Mr",
  BirthDate : "2000-01-01",
  HireDate : "2020-01-01",
  Notes : "Not available",
  PhotoPath : "http://google.com",
  Salary : 1000.0
}

insert Customers {
  @id: #GenBCons,
  CustomerID : "GENCU",
  CompanyName : "Gen Customer",
```

```
  ContactName : "Ana Lyse"
}

insert Orders {
  @id: #GenBOrder,
  OrderDate : "2020-11-01",
  Employee : #GenBEmployee,
  Customer : #GenBCons
}

insert Order_Details{
  @id: #OrderDetails,
  UnitPrice : 1.0,
  Quantity : 5,
  Discout : 0.0,
  Products : #GenBProduct,
  Orders : #GenBOrder
}

insert Order_Details{
  @id: #OrderDetails2,
  UnitPrice : 5.0,
  Quantity : 2,
  Discout : 0.5,
  Products : #GenBProduct2,
  Orders : #GenBOrder
}

update Orders o
where o.@id == #GenBOrder
set {Order_Details +: [#OrderDetails2,#OrderDetails]}

update Products p
where o.@id == #GenBProduct
set {Order_Details +: [#OrderDetails]}

update Products p
where o.@id == #GenBProduct2
set {Order_Details +: [#OrderDetails2]}

update Employees e
where e.@id == #GenBEmployee
set {Orders +: [#GenBOrder]}

update Customers c
where c.@id == #GenBCons
set {Orders +: [#GenBOrder]}

insert Employees {
  @id:#GenBEmployee2,
  LastName : "Generate",
  FirstName : "Mario",
  Title : "Mr",
  BirthDate : "1999-01-01",
  HireDate : "2020-06-01",
  Notes : "Not available",
  PhotoPath : "http://facebook.com/Mario",
```

```
  Salary : 2000.0,
  Employees_1 : #GenBEmployee
}

update Employees e
where e.@id == #GenBEmployee
set {Employees +: [#GenBEmployee2]}

insert Orders {
  @id : #GenBOrder2,
  OrderDate : "2020-11-01",
  Employee : #GenBEmployee,
  Customer : #GenBCons
}

insert Order_Details{
  @id: #OrderDetails3,
  UnitPrice : 1.0,
  Quantity : 5,
  Discout : 0.0,
  Products : #GenBProduct,
  Orders : #GenBOrder
}

insert Order_Details{
  @id: #OrderDetails4,
  UnitPrice : 5.0,
  Quantity : 2,
  Discout : 0.5,
  Products : #GenBProduct2,
  Orders : #GenBOrder
}

insert Order_Details{
  @id: #OrderDetails5,
  UnitPrice : 2.0,
  Quantity : 1,
  Discout : 1,
  Products : #GenBProduct3,
  Orders : #GenBOrder
}

update Orders o
where o.@id == #GenBOrder2
set {Order_Details +: [#OrderDetails3,#OrderDetails4,#OrderDetails5]}

delete Order_Details o
where o.@id == #OrderDetails5

update Products p
where p.@id == #GenBProduct
set {Order_Details +: [#OrderDetails3]}

update Products p
where p.@id == #GenBProduct2
set {Order_Details +: [#OrderDetails4]}

```

```
update Employees e
where e.@id == #GenBEmployee2
set {Orders +: [#GenBOrder2]}

delete Order_Details o
where o.@id == #OrderDetails4

update Customers c
where c.@id == #GenBCons
set {Orders +: [#GenBOrder2]}

update Products p
where p.ProductName =="Chai"
set {price :- 5}

delete Orders o
where o.@id == #GenBOrder

delete Customers c
where c.@id == #GenBCons

delete Employees e
where e.@id == #GenBEmployee
```

Appendix/Queries/balanced.tql

Bibliography

[1] P. Ameri et al. "NoWog: A Workload Generator for Database Performance Benchmarking". In: *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. 2016, pp. 666–673.

[2] Stefan Armbruster. *Welcome to the Dark Side: Neo4j Worst Practices (& How to Avoid Them)*. Dec. 2018. URL: `https://neo4j.com/blog/dark-side-neo4j-worst-practices/`.

[3] *BENCHMARK: meaning in the Cambridge English Dictionary*. URL: `https://dictionary.cambridge.org/dictionary/english/benchmark`.

[4] *BiDAWG documentation : Introduction and Overview*. URL: `https://bigdawg-documentation.readthedocs.io/en/latest/intro.html`.

[5] S. Ceesay, A. Barker, and B. Varghese. "Plug and play bench: Simplifying big data benchmarking using containers". In: *2017 IEEE International Conference on Big Data (Big Data)*. 2017, pp. 2821–2828.

[6] John Cieslewicz and Kenneth A. Ross. "Database Optimizations for Modern Hardware". In: *Proceedings of the IEEE* 96.5 (2008), pp. 863–878. DOI: `10.1109/JPROC.2008.917744`.

[7] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". In: *Commun. ACM* 13.6 (June 1970), pp. 377–387. ISSN: 0001-0782. DOI: `10.1145/362384.362685`. URL: `https://doi.org/10.1145/362384.362685`.

[8] Brian F. Cooper et al. "Benchmarking Cloud Serving Systems with YCSB". In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC '10. Indianapolis, Indiana, USA: Association for Computing Machinery, 2010, pp. 143–154. ISBN: 9781450300360. DOI: `10.1145/1807128.1807152`. URL: `https://doi.org/10.1145/1807128.1807152`.

[9] *Declarative language*. URL: `https://www.britannica.com/technology/declarative-language`.

[10] Y. Demchenko, C. de Laat, and P. Membrey. "Defining architecture components of the Big Data Ecosystem". In: *2014 International Conference on Collaboration Technologies and Systems (CTS)*. 2014, pp. 104–112.

[11] *Developer Apple: Human Interface Guidelines*. URL: `https://developer.apple.com/design/human-interface-guidelines/`.

[12] Juliana Freire, Philippe Bonnet, and Dennis Shasha. "Computational Reproducibility: State-of-the-Art, Challenges, and Database Research Opportunities". In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. SIGMOD '12. Scottsdale, Arizona, USA: Association for Computing Machinery, 2012, pp. 593–596. ISBN: 9781450312479. DOI: `10.1145/2213836.2213908`. URL: `https://doi.org/10.1145/2213836.2213908`.

[13] Vijay Gadepally et al. "The BigDAWG Polystore System and Architecture". In: (Sept. 2016).

[14] *Getting Started with Indexes*. URL: `https://mariadb.com/kb/en/getting-started-with-indexes/`.

[15] *Github: Core Workloads*. Accessed:2020-10-12. URL: `https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads`.

[16] *Graph Modeling Guidelines - Developer Guides*. URL: `https://neo4j.com/developer/guide-data-modeling/`.

[17] Theo Haerder and Andreas Reuter. "Principles of Transaction-Oriented Database Recovery". In: *ACM Comput. Surv.* 15.4 (Dec. 1983), pp. 287–317. ISSN: 0360-0300. DOI: `10.1145/289.291`. URL: `https://doi.org/10.1145/289.291`.

[18] Gernot Heiser. *Systems Benchmarking Crimes. (2010)*. 2010. URL: `https://www.cse.unsw.edu.au/~gernot/benchmarking-crimes.html`.

[19] Torsten Hoefler and Roberto Belli. "Scientific Benchmarking of Parallel Computing Systems: Twelve Ways to Tell the Masses When Reporting Performance Results". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450337236. DOI: `10.1145/2807591.2807644`. URL: `https://doi.org/10.1145/2807591.2807644`.

[20] *Index configuration - Operations Manual*. URL: `https://neo4j.com/docs/operations-manual/current/performance/index-configuration/#index-configuration`.

[21] *Introduction¶*. URL: `https://cassandra.apache.org/doc/latest/data_modeling/intro.html#what-is-data-modeling`.

[22] Sadhana J. Kamatkar et al. "Database Performance Tuning and Query Optimization". In: *Data Mining and Big Data*. Ed. by Ying Tan, Yuhui Shi, and Qirong Tang. Cham: Springer International Publishing, 2018, pp. 3–11. ISBN: 978-3-319-93803-5.

[23] Ingo Keep. *Performance Best Practices: Indexing: MongoDB Blog*. Feb. 2020. URL: `https://www.mongodb.com/blog/post/performance-best-practices-indexing`.

[24] Ingo Keep. *Performance Best Practices: Transactions and Read / Write Concerns: MongoDB Blog*. Feb. 2020. URL: `https://www.mongodb.com/blog/post/performance-best-practices-transactions-and-read--write-concerns`.

[25] Mat Keep. *Performance Best Practices: MongoDB Data Modeling and Memory Sizing: MongoDB Blog*. Jan. 2020. URL: `https://www.mongodb.com/blog/post/performance-best-practices-mongodb-data-modeling-and-memory-sizing`.

[26] N. Leavitt. "Will NoSQL Databases Live Up to Their Promise?" In: *Computer* 43.2 (2010), pp. 12–14.

[27] Gubichev A. et al. Leis V. Radke B. "Query optimization through the looking glass, and what we found running the Join Order Benchmark." In: *The VLDB Journal* 27.27 (2018), pp. 643–668. DOI: `10.1007/s00778-017-0480-7`. URL: `https://link.springer.com/article/10.1007/s00778-017-0480-7#citeas`.

[28] *Microsoft: Fuent Design System*. URL: `https://www.microsoft.com/design/fluent/#/`.

[29] Tamara Munzner. *Visualization Analysis and Design*. CRC Press, 2014.

[30] Todd Mytkowicz et al. "Producing Wrong Data without Doing Anything Obviously Wrong!" In: *SIGPLAN Not.* 44.3 (Mar. 2009), pp. 265–276. ISSN: 0362-1340. DOI: `10.1145/1508284.1508275`. URL: `https://doi.org/10.1145/1508284.1508275`.

[31] Raghunath Nambiar et al. "TPC Benchmark Roadmap 2012". In: *Selected Topics in Performance Evaluation and Benchmarking*. Ed. by Raghunath Nambiar and Meikel Poess. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–20. ISBN: 978-3-642-36727-4.

[32] Vijayakumar Nanjappan et al. "Chapter 1 - Big Data: A Classification of Acquisition and Generation Methods". In: *Big Data Analytics for Sensor-Network Collected Intelligence*. Ed. by Hui-Huang Hsu, Chuan-Yu Chang, and Ching-Hsien Hsu. Intelligent Data-Centric Systems. Academic Press, 2017, pp. 3–20. ISBN: 978-0-12-809393-1. DOI: https://doi.org/10.1016/B978-0-12-809393-1.00001-5. URL: https://www.sciencedirect.com/science/article/pii/B9780128093931000015.

[33] *Numeric vs String Fields*. URL: https://mariadb.com/kb/en/numeric-vs-string-fields/.

[34] P. Pyla R. Hartson. *The UX Book: Process and Guidelines for Ensuring a Quality User Experience*. Elsevier, 2012.

[35] Mark Raasveldt et al. "Fair Benchmarking Considered Difficult: Common Pitfalls In Database Performance Testing". In: *Proceedings of the Workshop on Testing Database Systems*. DBTest'18. Houston, TX, USA: Association for Computing Machinery, 2018. ISBN: 9781450358262. DOI: 10.1145/3209950.3209955. URL: https://doi.org/10.1145/3209950.3209955.

[36] *Relational Dataset Repository*. Accessed: 2020-10-07. URL: https://relational.fit.cvut.cz/dataset/Northwind.

[37] P.J. Sadalage and M. Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 2012.

[38] M. Stonebraker and U. Cetintemel. ""One size fits all": an idea whose time has come and gone". In: *21st International Conference on Data Engineering (ICDE'05)*. 2005, pp. 2–11.

[39] *TPC BENCHMARK C - Standard Specification*. Accessed:2020-10-12. URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf.

[40] *TPC-H is a Decision Support Benchmark*. Accessed:2020-10-12. URL: http://www.tpc.org/tpch/.

[41] *Tutorial: Import Relational Data Into Neo4j*. Accessed: 2020-10-07. URL: https://neo4j.com/developer/guide-importing-data-and-etl/.

[42] *TYPHON Overview*. URL: https://www.typhon-project.org/overview.

[43] *UML*. URL: https://www.uml.org/what-is-uml.htm.

[44] *VISUALISATION: meaning in the Cambridge English Dictionary*. URL: https://dictionary.cambridge.org/dictionary/english/visualization.

[45] *W3C Standards: Web Design and Applications*. URL: https://www.w3.org/standards/webdesign/.